Security Concept

Mar 09, 2021

INTRODUCTION

luca Sicherheitskonzept

luca unterstützt Veranstaltungs- und Restaurantbetreiber:innen bei der verpflichtenden Aufnahme der personenbezogenen Daten ihrer Gäste im Rahmen der Bekämpfung der Covid-19-Pandemie. Dabei legt *luca* besonderen Wert auf den Schutz dieser Daten: Die Daten können weder von Betreiber:innen, noch vom *luca*-System gelesen werden, sondern nur von den Gesundheitsämtern, in dem Falle dass eine Kontaktnachverfolgung wegen einer bestätigten Infektion eines Gastes notwendig wird.

Nutzer:innen von *luca* erzeugen beim Zutritt so genannte verschlüsselte Check-Ins mit ihrem Smartphone oder einem Schlüsselanhänger. Grundidee ist, dass die personenbezogenen Daten dabei von der *luca*-App der Nutzer:in mit einem Gesundheitsamt-Schlüssel verschlüsselt und im Rahmen des Check-Ins an der Betreiber:in geschickt werden. Die Betreiber:in verschlüsselt diese verschlüsselten Daten erneut und sendet sie, zusammen mit der Check-In-Zeit, an das *luca*-System. Hierdurch kann weder die Betreiber:in, noch das *luca*-System oder die Gesundheitsämter die personenbezogenen Daten der Nutzer:innen lesen.

Wird bei einer Nutzerin eine Infektion mit SARS-CoV2 festgestellt, sendet die *luca*-App der Infizierten alle Check-In-Zeiten zusammen mit der ID der Betreiber:innen an das *luca*-System. *luca* fordert von allen diesen Betreiber:innen eine Entschlüsselung der Daten der Gäste an, die sich in der betreffenden Zeit ebenfalls eingecheckt haben, d.h. *luca* erhält die mit dem Schlüssel des Gesundheitsamtes verschlüsselten Daten der Nutzer:innen. Diese Daten werden dann von *luca* an das zuständige Gesundheitsamt übermittelt, das hieraus die Daten entschlüsseln und damit die Nutzer:innen kontaktieren kann.

luca Security Concept

Due to the Covid-19 pandemic, restaurants, event venues and other public locations are required by law to collect the contact information of their guests. *luca* aims to simplify this process while putting an emphasis on personal data protection. Once a user tests positive for SARS-CoV2, only the public health authorities can access the end-user's personal data to conduct epidemiological contact tracing. Neither the owners of event venues nor the *luca* server can access the end-user's data at any time.

Upon entering a *luca*-enabled venue, a user creates a so-called "encrypted Check-In" using either their smartphone or a simple key fob. This Check-In is encrypted in a way that only the public health authorities can read it. The encrypted Check-In is then transferred to the venue owner via a QR code, where the Check-In is encrypted once again by the venue owner so that nobody can access the user's personal information at this stage. Now, the venue owner sends this double-encrypted Check-In to the *luca* server along with a time stamp.

In case a *luca* user tests positive for SARS-CoV2, the app sends the infected user's epidemiologically relevant check-ins along with the IDs of the affected venues to the *luca* server. *luca* then requests these venue owners to decrypt and upload the Check-In records of every user who visited their venue at the same time as the infected person. Note that all contact information is still encrypted by the public health authorities' key and readable by neither *luca* nor the venue owner. Once the public health authorities receive these Check-Ins, they can decrypt the data and contact *luca* users who might have been exposed to SARS-CoV2.

CHAPTER ONE

INTRODUCTION

1.1 About this Document

This document describes the security concepts of the *luca* system as well as the processes and cryptographic functions in technical detail. It also explains the guarantees *luca* provides to its users and how these guarantees are accomplished.

Both *luca* and this document are continuously improved and in active development. If you discover any issues with the concepts in this document or any mismatch between the document and *luca*'s behaviour, please contact us directly at security@luca-app.de for responsible disclosure.

We greatly appreciate your feedback.

1.2 Contributors

This document is owned by culture4life GmbH, which is also responsible for the development of *luca*. It is continuously developed and reviewed in cooperation with security experts and partners such as neXenio GmbH and Fraunhofer AISEC.

1.3 Guarantees Provided by *luca*

luca's main goal is to protect guests' personal data. The technical description of the guarantees *luca* aims to provide to its users can be found in the chapter *Security Objectives*.

In contrast to the paper-based approach to collecting contact data at restaurants and other public venues, *luca* is designed to prevent the venue's staff, *luca* itself and other 3rd parties from accessing this data. Public health authorities (i.e. "Gesundheitsämter") are the only entity that can access the relevant personal data of guests to conduct contact tracing of users who have been potentially exposed to SARS-CoV2. Similar to the traditional paper-based contact data collection, the health authorities need the venue owner's consent to access this information.

luca aims to underpin all *security and data protection objectives and guarantees* with cryptographic protocols wherever feasible. This document describes the current implementation status of the *luca* system and provides security considerations where some aspects of these guarantees are not yet fully met.

Please also note the *planned improvements*.

1.4 Overview

The remainder of this document is divided into six sections. The first section, “System Overview”, explains the important components, assets, security objectives and cryptographic secrets in the system. Section two describes how different actors are onboarded and registered in the system. Sections three and four describe the parts of the system most visible to our users: the various ways users can check-in at *luca* venues. Finally, the section “Contact Tracing” describes how public health authorities can use *luca* to identify chains of infection with the explicit consent of *luca* venues. The appendix contains technical details about the cryptography used in *luca* and improvements scheduled for the near future.

CHAPTER

TWO

ACTORS AND COMPONENTS

2.1 Actors

Guest A person that is required to securely provide their *Contact Data* to the Luca system before entering a venue and later (on infection) submit their location history (*Check-In History*) to the *Health Department*.

Technically the Guest can be in one of three roles: *Uninfected Guest*, *Traced Guest* or *Infected Guest*. Depending on their role, the security guarantees provided for the Guest change. For example, no component in the system other than the *Guest App* will ever learn an *Uninfected Guest*'s *Contact Data*. In contrast, the *Contact Data* of *Traced Guest*s is made available to the *Health Department*.

Uninfected Guest The default role of a Guest. Neither the *Check-In History* nor *Contact Data* has been shared with the *Health Department*.

Traced Guest A Guest who is part of a *Contact Tracing Process*. This Guest's *Contact Data* is revealed to the *Health Department*.

Infected Guest A Guest who is suspected of being infected with Sars-CoV2 and has consented to sharing their *Check-In History* with the *Health Department*.

Health Department A local Health Department responsible for identification of contact persons. This term is used synonymously for an employee that represents this Health Department.

Venue Owner A private person or business owner/manager of a venue that has Guests and uses *luca* to trace contact information.

Scanner Operator The person who operates the *Scanner Frontend* at a venue.

Luca Service Operator culture4life GmbH as the creator and operator of the Luca system as a whole, their backend services, phone and web applications. The Luca Service Operator has unrestricted access to the *Luca Server* database.

Trusted 3rd Party A person or institution that is not affiliated with *luca*, its developers or operators. A trusted 3rd party is required to perform vital initialization steps regarding *luca's* system setup. Note that different mentions of *Trusted 3rd Party* throughout the document can refer to different institutions.

2.2 Components

Guest App The Luca Guest App is the interface for Guests. Guests enter their *Contact Data* in the app and use the app to check-in at several locations without re-entering their contact details.

Health Department Frontend Enables the *Health Department* to trace infection cases and contact potential contact persons.

Venue Owner Frontend The frontend for the managers/organizers of venues/locations/restaurants. Here, a professional or private user can create locations or events which will enable them to check-in Guests.

Luca Server Stores encrypted *Check-In*s and *Contact Data* and centrally orchestrates the other technical components. The Luca Server is never in possession of personal *Contact Data* in plain text.

Scanner Frontend A web app that is used by the organizer or their employees to scan the QR codes produced by the *Guest App* to check-in Guests.

Web-Check-In Frontend The Web-Check-In frontend enables venues to let Guests enter their information directly on-site on tablet device or something similar. This is useful if Guests are not users of the *Guest App*.

Badge Personalization Frontend A web application to encrypt and store *Contact Data* for *Guest*'s that are wishing to use a *Badge* to check-in at Luca locations.

Badge Generator Generates printable QR codes to be used by people without smartphones to allow check-ins at *luca* venues.

Badge A printable QR code in the form of small badge to allow people without a smartphone to check-in at *luca* locations.

Email Service Provider Used to *Venue Owner*s and *Health Department*s.

SMS Service Provider Used to validate a Guest's phone number upon entering *Contact Data* in the *Guest App*.

Component Diagram

```
<IPython.core.display.SVG object>
```

CHAPTER

THREE

ASSETS

Check-In The central artifact that enables the *Health Department* to contact *Traced Guest*s. It is created with each *check-in process* and encodes the information that a Guest is located at a specific location at a specific time. However, the information which Guest the Check-In belongs to (specifically, their *Contact Data*) is only made available to the Health Department during *Tracing the Check-In History of an Infected Guest* and is never available to any other actor in the system.

Each Check-In contains the following information:

- the venue where the Check-In was created
- the check-in time
- the check-out time
- a *trace ID* that can be connected to an *Infected Guest* during contact tracing
- the encrypted *contact data reference* that can be used by the Health Department to contact *Traced Guest*s

Check-In History A collection of multiple *Check-In*s that all belong to the same Guest. This asset is only made availabe to the *Health Department* when an *Infected Guest* decides to share it with the Health Department to aid in *Tracing the Check-In History of an Infected Guest*. It allows the Health Department to reconstruct all venues the Guest has visited during the epidemiologically relevant timespan.

Contact Data The personal contact data that is entered by the Guest into the Guest App upon registration. It contains the following information[1]:

- first and last name
- full address (street, street number, city, postal code)
- phone number
- email address

Health Department Information The meta data collected by the *Luca Server* about a Health Department during *Health Department Registration*. The following information is stored:

- name of the Health Department
- for each configured employee:
  - first and last name
  - email address
  - phone number (if provided)

Venue Information The meta data about a venue. The following information is stored:

- name of the venue
- name, email address and phone number contact person for this venue

[1] *luca* is required to collect this information in order to comply with the German Infektionsschutzgesetz law and all federal states' implementations of the law.

- full address of the venue
- geo-coordinates of the venue
- configured Check-In radius
- configured number of tables

Scanner Information The meta data of a QR code scanner tied to a venue. *Venue Owner*s can create any number of unique scanners for their venue. The *Luca Server* maintains the following information for each scanner:

- *scanner ID*
- *venue ID*
- a plaintext name of the scanner
- the public key of the *venue keypair*

CHAPTER

FOUR

SECURITY OBJECTIVES

luca provides the following guarantees to the respective actors in the system:

4.1 List of Objectives

O1. An Uninfected Guest's Contact Data is known only to their Guest App

The Guest's personal data is undisclosed as long as they didn't test positive (and become an *Infected Guest*) or show up in a tracing process by a *Health Department* (and become a *Traced Guest*).

O2. An Uninfected Guest's Check-Ins cannot be associated to the Guest

Individual *Check-In*s of an *Uninfected Guest* are not disclosed. Only when a Check-In shows up in a tracing process (making the Guest a *Traced Guest*), is this particular Check-In disclosed to the *Health Department*.

Naturally, the Guest App itself may have knowledge about the Check-Ins.

O3. An Uninfected or Traced Guest's Check-Ins cannot be associated to each other

The entire *Check-In History* of a Guest is disclosed to the *Health Department* if, and only if, the Guest tested positive and explicitly consents to the tracing (making them an *Infected Guest*). Thus, not even an anonymous *Check-In History* can be generated.

Note that the Guest App may keep a local history of Check-Ins.

O4. An Infected Guest's Check-In History is disclosed to the Health Department only after their consent

Even if a Guest tested positive and is in contact with the *Health Department*, they can decide not to share their *Check-In History*.

O5. The Health Department learns only the relevant part of the Infected Guest’s Check-In History

The *Health Department* only learns the epidemiologically relevant part of a Guest’s *Check-In History*.

O6. Traced Guest’s Contact Data is disclosed to the Health Department only after Venue Owners’ consent

This requirement is meant to mitigate illicit disclosure of arbitrary Guests’ contact information by the authorities.

CHAPTER

FIVE

SECRETS AND IDENTIFIERS

5.1 System-wide List of Secrets

daily keypair The keypair whose public key is used by the *Guest App* to encrypt the secret part of the *Check-In* data. Its private key is used by a *Health Department* during the process of *Contact Tracing*.

The keypair's public key is signed using the *HDSKP* and stored on the *Luca Server*. Its private key is encrypted for each registered Health Department's *HDEKP*. The encrypted private keys are stored on the *Luca Server*.

The daily keypair's life cycle and usage is detailed in the chapter *Daily Keypair Rotation*.

badge keypair The keypair that encrypts *contact data reference*s for static *Badge*s. The public key is used exclusively by a *Trusted 3rd Party* during *the generation of static Badges*. Its private key is owned by the Health Department and is used to decrypt *Check-In*s created using a static Badge.

badge attestation keypair This keypair signs static *Badge*s during *their generation*. Its private key is kept in the *Luca Server* and is used via an authenticated API endpoint by the *Badge Generator*. The *Scanner Frontend* uses the public key to verify that a presented *Badge* is valid and registered with the *Luca Server*.

data secret A secret cryptographic seed which is used to derive both the *data encryption key* and the *data authentication key*. This seed is encrypted twice before being sent to the *Luca Server* during *Check-In* and ultimately protects the Guest's *Contact Data*. It is stored locally in the *Guest App*.

data encryption key A symmetric key derived from the *data secret*, used to encrypt the *Contact Data*.

data authentication key A symmetric key derived from the *data secret* during *Guest Registration*. It is used to authenticate the Guest's *Contact Data* and *Check-In*s. The *data authentication key* is stored encrypted on the *Luca Server* as a part of the *encrypted guest data*.

guest keypair An asymmetric keypair created during the *Guest Registration*.

The keypair's private key is used to sign the *encrypted guest data* and *guest data transfer object*. The public key is uploaded to the *Luca Server*.

HDEKP The "Health Department Encryption Keypair" is used to encrypt the *daily keypair*'s private key. Each Health Department has their own HDEKP.

The public of this keypair is signed using the *HDSKP* and stored on the Luca Server. The private key is stored locally at the Health Department.

HDSKP The "Health Department Signing Keypair" is used to authenticate the *HDEKP*. Each Health Department has their own HDSKP.

Health Department Certificate A certificate that identifies a *Health Department*. It is used to authenticate to the *Health Department Frontend*.

This certificate is created in a manual process by the *Luca Service Operator* and signed by an external, trusted Certificate Authority.

tracing secret A randomly generated seed used to derive *trace ID*s when *checking in using the Guest App*. It is stored locally on the *Guest App* until it is shared with the *Health Department* during *contact tracing*.

Moreover, the tracing secret is rotated on a regular basis in order to limit the number of *trace ID*s that can be reconstruced when the secret is shared.

tracing TAN The tracing TAN (Transaction Authentication Number) is a human readable code that is used during the process of *Contact Tracing*. By requesting a TAN from the *Luca Server* and communicating it to the *Health Department* an *Infected Guest* grants the Health Department access to their *Contact Data*.

Note: This TAN is not to be confused with the *verification TAN*, which is involved in the *Guest Registration* process to verify the Guest's phone number.

venue keypair An asymmetric keypair generated locally in the *Venue Owner Frontend* upon *Venue Registration*. The keypair's public key is used by the *Scanner Frontend* to add the outer layer of encryption to the *contact data reference* (which is already encrypted for the *daily keypair*) during *Guest Check-In*. Its private key is stored locally.

verification TAN The verification TAN (Transaction Authentication Number) is a human readable code that is used to verify the Guest's phone number during *Guest Registration*.

badge serial number The 12-digit serial number that is printed on the flip-side of each *Badge*. A 56-bit random number that acts as a seed to derive all secrets associated with the *Badge* and encoded into the *Badge*'s QR code.

5.2 Glossary

user ID A unique identifier for the Guest in the Luca system. It indexes the *encrypted guest data* and is also used to derive *trace ID*s during *Guest Check-In*.

trace ID An opaque identifier derived from a Guest's *user ID* and *tracing secret* during *Guest Check-In*. It is used to identify *Check-In*s by an *Infected Guest* after that Guest shared their *tracing secret* with the *Health Department*.

venue ID An unique identifier for a venue registered in the Luca system. The venue ID is linked to the *Venue Information* stored by the *Luca Server*.

scanner ID An unique identifier for an instance of a *Scanner Information* associated with a specific venue. Given the scanner ID the *Scanner Frontend* can start performing *Check-In*s for the associated venue.

daily keypair ID An identifier for the *daily keypair*.

verification tag A tag used to verify the authenticity of the *contact data reference*.

encrypted guest data This object contains the *Contact Data* and *data authentication key*. It is encrypted with the *data encryption key*, signed with the *guest keypair* and uploaded to the *Luca Server* during *Guest Registration*.

guest data transfer object This object contains an *Infected Guest*'s *tracing secret*s, *user ID* and *data secret*. During *Tracing the Check-In History of an Infected Guest* the *Guest App* encrypts the *guest data transfer object* for the *daily keypair* and shares it (via the *Luca Server*) with the *Health Department*.

contact data reference The *contact data reference* combines the *user ID*, the *data secret* and a *verification tag*. Encrypted with both the *daily keypair* and the *venue keypair* it is included in each *Check-In* during *Guest Check-In*.

CHAPTER

SIX

OVERVIEW OF PROCESSES

luca's primary goal is to automate the identification of contact persons for venues, restaurants and locations and ease the *Health Department*'s identification of possible contact persons of infected persons.

The table below provides a high-level overview of the involved processes to achieve this goal. The chapters in this part explain each process in detail.

Process Name	Purpose
Guest Registration	Set up a smartphone to use the *Guest App* in order to check-in at venues using *luca*.
Venue Registration	Register an event or a venue with the Luca system and enable it to check-in Guests via *luca*.
Health Department Registration	Onboard a *Health Department* to the Luca system.
Daily Public Key Rotation	Regular rotation of the *daily keypair*.
Check-In via Mobile Phone App	Transmit encrypted contact information to the *Health Department* so it can be used for *Tracing the Check-In History of an Infected Guest*.
Tracing the Check-In History of an Infected Guest	Retrieve a *Traced Guest*'s *Contact Data* in case of an infection.
Generation of Static Badges, *Badge Personalization*, *Badge Check-In*	Set up a Static QR code *Badge*, personalize it with the *Contact Data* of its owner and use it to check-in at venues without using a smartphone.

CHAPTER

SEVEN

VENUE REGISTRATION

Professional *Venue Owner*s can register their venue with the *luca* system via a web application. The venue can then be managed via a web interface in order to set up individual *Scanner Frontend*s and to configure other venue-specific parameters (for example auto checkout behavior).

7.1 Overview

Participants

- *Luca Server*
- *Venue Owner*
- *Venue Owner Frontend*

Assets

- *Venue Information*

Preconditions

- the venue is not registered

Postconditions

- the *venue keypair* is available locally to the *Venue Owner Frontend*
- the *Venue Information* is stored on the *Luca Server*

7.2 Secrets

The following *secrets* are involved in this process:

Secret	Use / Purpose	Location
venue keypair	Encrypt the *contact data reference* of Guests during *check-in* and decrypt it during *Tracing the Check-In History of an Infected Guest*.	Both the public and private key are stored locally by the *Venue Owner Frontend*. The public key is shared with *Scanner Frontend*s when they are set up.

7.3 Process

To initiate the process the *Venue Owner* registers with their email address and a password. They enter further information, such as the name of the venue and their contact information in the *Venue Owner Frontend* (see *Venue Information* for the complete list of the data collected).

Subsequently, the Venue Owner Frontend generates the *venue keypair*. Both the public and private key are stored locally. The keypair's public key is used to set up new *Scanner Frontend*s, which utilize it to encrypt Guests' *contact data reference* during *Check-In via Mobile Phone App*. The keypair's private key is needed by the Venue Owner Frontend in order to lift this encryption when assisting a *Health Department* in the process of *Tracing the Check-In History of an Infected Guest*.

7.4 Security Considerations

Authenticity of the Venue Keypair's Public Key

As the *Venue Owner* holds no certificate with which they could sign the public key of the *venue keypair* there is no secure way to validate its authenticity when it is used in the check-in process. This affects both the *Check-In via Mobile Phone App* and the *Check-In via a Printed QR Code*.

It is therefore important that the public key is transmitted to the *Scanner Frontend* on a secure out-of-band channel (specifically, not the *Luca Server*).

Prospectively, this will be implemented by attaching the *venue keypair*'s public key to the fragment component of the link to the *Scanner Frontend*, which is created in the *Venue Owner Frontend*. For printed QR codes for self Check-In the public key will be part of the QR code.

Note that the impact of this only affects the outer layer of the *contact data reference*'s encryption. It is still encrypted with the *daily keypair* and thus only accessible for the *Health Department*.

Sensitivity of the Venue Keypair

The *venue keypair*'s private key must not be lost or made accessible to third parties. Hence, organizational measures are taken to specifically inform the *Venue Owner* that special care must be taken when dealing with this key.

CHAPTER

EIGHT

HEALTH DEPARTMENT REGISTRATION

luca helps *Health Department*s to trace contact persons and identify infection clusters. In order to participate in the system Health Departments need to be registered and onboarded first.

8.1 Overview

Participants

- *Luca Server*
- *Health Department*
- *Health Department Frontend*

Assets

- *Health Department Information*

Preconditions

- the *Health Department* is not onboarded

Postconditions

- the *Health Department* has received a login certificate from the *Luca Service Operator* (out-of-band)
- an admin user for the *Health Department* has been registered
- the *Health Department Information* is stored on the *Luca Server*
- the Health Department's *HDEKP* and *HDSKP*'s public keys are stored on the *Luca Server*; the private keys are stored locally at the Health Department
- relevant *daily keypair*s have been re-encrypted by an existing Health Department

8.2 Secrets

The following *secrets* are involved in this process:

Secret	Use / Purpose	Location
HDEKP	Encrypt/decrypt the *daily keypair*.	The private key is stored locally on the device that runs the *Health Department Frontend*. The public key is stored on the *Luca Server*.
HDSKP	Sign the *daily keypair* during *Daily Public Key Rotation*.	The private key is stored locally on the device that runs the *Health Department Frontend*. The public key is stored on the *Luca Server*.
Health Department Certificate	Authenticate to the *Health Department Frontend*.	Stored locally on devices that run the *Health Department Frontend*.

8.3 Process

In order to be onboarded to *luca* the *Health Department* contacts the *Luca Service Operator*. From them the Health Department receives the *Health Department Certificate*. The Luca Service Operator also helps to provide the *Health Department Information* to the *Luca Server* and to set up an admin user account for one of the Health Department's employees. The admin user can now access the *Health Department Frontend* using the certificate and the credentials for their user account.

When the admin user logs in for the first time the *Health Department Frontend* automatically generates two keypairs: the *HDEKP* and the *HDSKP*. These keypairs are required for the *Daily Key Rotation Process*. Please refer to that chapter for more details about the keypairs' usage. Both keypairs' public keys are uploaded to the *Luca Server*. Their private keys are stored locally.

Re-Encryption of the Daily Keypair

In the final step of the onboarding process all recent (epidemiologically relevant) *daily keypair*s need to be re-encrypted for the new *Health Department*. This is necessary in order for the new Health Department to be able to decrypt existing *daily keypair*s with its *HDEKP*. The re-encryption process is triggered automatically and carried out by any other Health Department that is currently logged in to the *Health Department Frontend* as follows:

- fetch all Health Departments' *HDEKP* public keys (including the new Health Department's recently created key)
- download all relevant *daily keypair*s
- decrypt them using its own *HDEKP*'s private key
- encrypt them for all other Health Departments' *HDEKP*s
- upload them back to the *Luca Server*

This process is very similar to the *rotation of the daily keypair*. Please refer to that chapter for further details.

Adding Further (Non-Admin) Employees

The admin user can create further user accounts that do not have administrative access in the *Health Department Frontend*. Like the admin user, those users can authenticate to the Health Department Frontend using their individual credentials and the *Health Department Certificate* and use it for contact tracing.

CHAPTER

NINE

DAILY KEYPAIR ROTATION

9.1 Overview

Participants

- *Health Department*
- *Health Department Frontend*
- *Luca Server*

Assets

- None

Preconditions

- the *Health Department* is *registered with the Luca system*
- the current *daily keypair* is older than its rotation threshold

Postconditions

- a fresh *daily keypair* is generated and published to the *Luca Server*
  - *Guest App*s use the new public key for *Check-In*s
  - all *Health Department*s have access to the new private key for *contact tracing*

9.2 Secrets

The following *secrets* are involved in this process:

Secret	Use / Purpose	Location
daily keypair	*Guest App*s use the *daily keypair*'s public key to encrypt their *contact data reference* for every *Check-In*. The *daily keypair* is rotated frequently to minimize potential misuse.	Private key is accessible to all *Health Departments*
HDSKP	New *daily keypair* public keys are signed by the *Health Department*'s private key so that *Guest Apps* can validate the public key's authenticity.	Every *Health Department* maintains their own *HDSKP* locally. Public keys are distributed via the *Luca Server*[1].
HDEKP	New *daily keypair* private keys are encrypted for each *Health Department* via their associated *HDEKP*.	Every *Health Department* maintains their own *HDEKP* locally. Public keys are distributed via the *Luca Server*[2].

9.3 Daily Public Key Rotation

For every *Check-In* the *Guest App* encrypts a *contact data reference* with the *daily keypair*. To mitigate the impact of any single compromised key *luca* rotates the *daily keypair* frequently.

The rotation will be performed by any *Health Department* that logs in after the last *daily keypair* expired. The private key is encrypted and shared by all participating *Health Department*s using their associated *HDEKP*s (Health Department Encryption Key Pair) via the *Luca Server*. The public key (and its creation date) are signed with the *HDSKP* (Health Department Signing Key Pair) and distributed to all *Guest App*s via the *Luca Server*. This effectively replaces the old *daily keypair*. All described cryptographic actions are performed in the *Health Department Frontend*, the *Luca Server* never learns the *daily keypair* private key in plaintext form.

Measures are taken to solve race conditions if multiple *Health Department*s try to perform the key rotation simultaneously. Eventually, all *Health Department*s share the knowledge of the new *daily keypair* and are ready to decipher *Contact Data* of *Check-In*s performed on that day.

Rotation Process

```
<IPython.core.display.SVG object>
```

Key Destruction

Private keys of *daily keypair*s that are older than the epidemiologically relevant time span (specifically, four weeks) can be destroyed. The *Luca Server* removes all such encrypted private keys for all *Health Department*s. Furthermore, the *Health Department Frontend* removes all locally stored copies of such private keys.

Security Considerations

Authenticity of HDSKP and HDEKP

Each *Health Department* owns a pair of keypairs, namely *HDSKP* and *HDEKP*. Those keypairs are used to authenticate and distribute newly generated *daily keypair*s. Both *HDSKP* and *HDEKP* are generated in the *Health Department Frontend* during *the registration process* and remain known exclusively to the respective *Health Department*. In a *future version of _luca_*, we plan to certify the public keys of *HDSKP* and *HDEKP* by an independent trusted certificate authority to further strengthen their authenticity guarantees.

---

[1] Currently, the *Health Department*s provide verbatim public keys as HDSKP/HDEKP, only. A future version of *luca* will also provide means to verify the authenticity of those public keys against a trusted third party.

CHAPTER

TEN

GUEST REGISTRATION

In this process a new *Guest* registers to the Luca system. This process is required for Guests using the *Guest App*. During this process, local secrets are created, the Guest enters their *Contact Data* and identifiers and encrypted data are sent to the *Luca Server*.

10.1 Overview

Participants

- *Guest*
- *Guest App*
- *Luca Server*

Assets

- *Contact Data*

Preconditions

- the Guest is not registered
- the Guest App is installed

Postconditions

- the Guest has a *user ID* and a *guest keypair*
- the *Guest App* has stored the private key material and secrets
- the *Luca Server* has stored the encrypted *Contact Data*

10.2 Secrets

The following *secrets* are involved in this process:

Secret	Use / Purpose	Location
data secret	A secret "seed"[1] which is used to derive both the *data encryption key* and the *data authentication key*. This seed is the secret that is encrypted twice before being sent to the *Luca Server* during *Check-In* and ultimately protects the Guest's *Contact Data*.	Securely[2] stored locally on the mobile device
data encryption key	A symmetric key derived from the *data secret*, used to encrypt the *Contact Data*.	Not stored
data authentication key	A symmetric key derived from the *data secret*, used to authenticate the *Contact Data*. It is also used to authenticate *Check-In*s.	Not stored
tracing secret	Used by the *Guest App* to generate *trace ID*s during *Check-In* and (after the *Guest* granted access to it) by the *Health Department* for *contact tracing*.	Securely stored locally on the mobile device
guest keypair	A pair of public and private key that is used to authenticate both new and updated *encrypted guest data*.	The public key is stored on the *Luca Server*. The private part is securely stored locally on the mobile device.

10.3 Process

The diagram below provides an overview to the complete process.

```
<IPython.core.display.SVG object>
```

Creating the Secrets

On initial startup the *Guest App* generates the following secrets:

- *data secret* as 16 bytes of random data
- *tracing secret* as 16 bytes of random data
- *guest keypair* as an EC `secp256r1` keypair

It stores this data[2]. The Guest App then derives two keys from *data secret* as follows:

- *data encryption key* as `SHA256(data secret || 0x01)`, truncated to 16 bytes
- *data authentication key* as `SHA256(data secret || 0x02)`

The derived keys are used in the remainder of the registration process but are not persisted on the device.

[1] The reason for deriving the two secrets from a seed rather than creating both of them at random is the limited "storage capacity" of the QR code during *Check-In*.

[2] All secrets stored on the device are protected using the respective OS' native credential storage mechanism. Specifically, the Android keystore system on Android and the iOS Keychain Services on iOS.

Verifying the Contact Data

Upon first launch of the *Guest App* the *Guest* provides their *Contact Data* to the App.

The App then verifies the Guest's phone number. For that, the phone number the Guest entered is sent to the *Luca Server*, which then creates a challenge. Using an external *SMS Service Provider* a *verification TAN* is sent to that phone number either as an SMS or as a voice call. After entering the TAN in the App it is verified by the *Luca Server*. The *Luca Server* keeps no record of the Guest's phone number after that[3].

Encrypting the Contact Data

If the TAN verification is successful, the *Guest App* creates and signs *encrypted guest data* as follows:

```
# pseudocode

iv = random_bytes(16)

encrypted_guest_data = AES_128(contact_data + data_authentication_key,
                               key=data_encryption_key,
                               mode=CTR,
                               iv=iv)

guest_data_mac = HMAC(encrypted_guest_data,
                      key=data_authentication_key)

guest_data_signature = guest_keypair.private.sign(encrypted_guest_data +
                                                  data_authentication_key +
                                                  iv)
```

The artifacts created here are uploaded to the *Luca Server* in the next step.

Registering to the Luca Server

The *Guest App* sends the following data to the *Luca Server*:

- the *encrypted guest data*
- the `IV` used in the encryption
- the `guest data mac`
- the `guest data signature`
- the public key of the *guest keypair*

The *Luca Server* returns the Guest's *user ID*. In the end of the process the two participants have stored the following data:

Guest App

- *user ID*
- *data secret*
- *tracing secret*
- *guest keypair* (public and private key)

Luca Server

- *user ID*
- the *guest keypair*'s public key

[3] According to the German "Telekommunikationsgesetz" the SMS Service Provider is legally required to store the messages for 90 days.

- the *encrypted guest data*

Updating the Contact Data

Guests should be able to update their *Contact Data*. This is needed, for example, in case their address or phone number changes. Whenever a Guest changes their *Contact Data* in the *Guest App*, the App creates a new *encrypted guest data* package by repeating the steps described in the Section *Encrypting the Contact Data*. Using the *user ID* retrieved during the initial registration, the App uploads the following data to the *Luca Server*:

- the *encrypted guest data*
- the `IV` used in the encryption
- the `guest data mac`
- the `guest data signature`

The *Luca Server* verifies that the Guest is authorized to update the data by checking the signature with the already present *guest keypair*'s public key.

Rotating the Tracing Secret

The *tracing secret* is used by the *Guest App* to generate the *trace ID*s for *Check-In*s. Whoever knows a *tracing secret* (and the respective *user ID*) can calculate all *trace ID*s the App has derived from this secret and thus reconstruct a part of the Guest's *Check-In History*.

This is desired, of course, when an *Infected Guest* consents to revealing their *tracing secret* to the *Health Department* during *Contact Tracing*. However, the Health Department should only learn the epidemiologically relevant part of the Guest's Check-In History (cf. *Security Objective O5*).

To guarantee this, the *Guest App* rotates the *tracing secret* once a day. If the Guest is infected, the App transfers all recent, epidemiologically relevant, *tracing secret*s to the *Health Department*. As a result, the Health Department can only reconstruct that part of the *Check-In History* which has been created based on the shared *tracing secret*s.

CHAPTER

ELEVEN

CHECK-IN VIA MOBILE PHONE APP

11.1 Overview

Participants

- *Guest*
- *Guest App*
- *Scanner Operator*
- *Scanner Frontend*
- *Luca Server*

Assets

- *Check-In*
- *Venue Information*

Preconditions

- the *Guest App* is installed
- the *Guest is registered*
- the *Venue Owner is registered*
- the *Scanner Frontend* is ready to scan

Postconditions

- the *Guest* has created a *Check-In* at the *Venue Owner*'s venue
- the *Guest* has checked-out of the venue at a later point in time
- the *Luca Server* has an encrypted record of the above *Check-In* Event
- the *Guest App* has noted the visited location in its local *Check-In History*

11.2 Secrets

The following *secrets* are involved in this process:

Secret	Use / Purpose	Location
data secret	The *Guest*'s secret seed to derive both the *data encryption key* and the *data authentication key*. This seed will be encrypted for the *Health Department* in this process before being transported to the *Scanner Frontend* via a QR code and protects the Guest's *Contact Data* stored on the *Luca Server*.	Securely stored locally on the mobile device (see *Guest Registration* for further details)
data authentication key	A symmetric key derived from the *data secret*, used to bind the *Check-In* data to the current time stamp of the Check-In event.	Not stored
tracing secret	Used to generate an anonymous *trace ID* to facilitate *contact tracing* by the *Health Department* (after the *Guest* granted access to the *tracing secret* – rendering them an *Infected Guest*).	Securely stored locally on the mobile device
daily keypair	Used to encrypt the above-mentioned *data secret* on the *Guest*'s mobile device before transferring it to the *Scanner Frontend* via a QR code.	The public key is obtained from the *Luca Server*[1]. The private key is known to all *Health Department*s (see *Daily Public Key Rotation* for further details).

11.3 Process

This describes how *Guest*s use *luca's Guest App* to generate so-called *Check-In*s at specific venues. For that, *Venue Owner*s deploy *Scanner Frontend*s that read QR codes generated by the *Guest App*. Note that there are other ways a *Guest* might check-in to a venue: Please refer to *Badge Check-In* and *Check-In via a Printed QR Code* for further details.

Check-Ins can be used at a later time by *Health Department*s to reconstruct an *Infected Guest*'s *Check-In History* (given that the *Infected Guest* has given their consent). Check-Ins of other Guests can be associated with the *Infected Guest*'s *Check-In History* to allow for *Tracing the Check-In History of an Infected Guest*.

In any case the Check-In data that is transferred and stored on the *Luca Server* does not reveal information about the Guest's identity (*O1*, *O2*). Neither does the Luca system learn about a Guest's habits (*O3*).

```
<IPython.core.display.SVG object>
```

Preparation

Before generating any QR codes to perform Check-Ins the *Guest App* will fetch the latest *daily keypair* public key from the *Luca Server* (see *Daily Public Key Rotation*). The provided public key comes with a reference to the issuing Health Department, a creation timestamp and a signature by a *Health Department*'s *HDSKP* certificate. The Guest App must validate this signature before encrypting anything with the fetched public key. Validation of the signature is subject to the planned improvement *Certification of Health Department Keypairs*. Furthermore, keys that are older than seven days are not considered valid anymore.

[1] The provided public key of the *daily keypair* is signed by a *Health Department* using their *HDSKP*. This signature is provided by the *Luca Server* along with said public key.

Scanner Check-In

QR codes generated by the *Guest App* are valid for a short period of time and the whole generation process described below is repeated every minute. Each *trace ID* is generated as HMAC-SHA256 of the Guest's *user ID* and a current quantized timestamp (clamped to the latest full minute) as `data` and the *tracing secret* as `key`. The resulting value is truncated to the first 16 bytes. Subsequently, the Guest App *asymmetrically encrypts* the Guest's *user ID* and the *data secret* for the *daily keypair*. The IV is defined as the first 16 bytes of the ephemeral public key used in the DLIES. The Guest App then calculates a *verification tag* as HMAC-SHA256 of the timestamp and the encrypted data as `data` and the *data authentication key* as key, truncated to the first 8 bytes. A four-byte checksum (truncated SHA256) of all the previously generated data blob is appended as an integrity check to detect faulty QR code reads[2].

QR Code Generation and Check-In

The app generates a new QR code every minute, for each code the app generates the following:

```
timestamp        = UNIX timestamp rounded down to the last full minute (little␣
↪endian encoding)
trace_id         = HMAC-SHA256(user_id || timestamp, tracing_secret)  # truncated␣
↪to 16 bytes
ephemeral_keys   = a new secp256r1 key pair (for DLIES with the daily public key)
dh_key           = ECDH(ephemeral_keys.private, daily_keypair.public)
enc_key          = SHA256(dh_key || 0x01)  # truncated to 16 bytes
iv               = ephemeral_keys.public   # truncated to 16 bytes
enc_data         = AES-128-CTR(userId || data_secret, enc_key, iv)
verification_tag = HMAC-SHA256(timestamp || enc_data, data_authentication_key)
```

Security Considerations

`trace_id`

The *trace ID* (`trace_id`) depends on the *user ID* (`user_id`) of the Guest, the current quantized timestamp and the Guest's *tracing secret*. Hence, all *trace IDs* for any given minute can be calculated given the *user ID* and the *tracing secret* (which is stored securely inside the *Guest App*). Without the tracing secret, the Guest's trace IDs can neither be linked to (a) the Guest themselves (fulfilling *O2*) nor (b) to other trace IDs of the same Guest (fulfilling *O3*).

If tested positive for Sars-CoV2 a *Guest* may consent to sharing their *tracing secret* with the *Health Department* (rendering them an *Infected Guest*). This facilitates the Health Department to trace the Infected Guest's *Check-In History* (fulfilling *O4*). See *Tracing the Check-In History of an Infected Guest* for further details.

To restrict the disclosed time interval of the *Infected Guest*'s *Check-In History* the Guest App regularly changes the *tracing secret* (see *Rotating the Tracing Secret*). The Guest App shares only the *tracing secret*s that were valid in an epidemiologically relevant time frame (about two weeks) with the *Health Department* (fulfilling *O5*).

[2] The QR code standard already includes an error correction mechanism. However, some dedicated QR code scanner hardware acts as keyboard input device to forward QR code data to the *luca* web application. As this data transfer appears to be error prone, we added checksumming on application level as well.

verification_tag

The encrypted data enc_data is not authenticated as it would usually be the case (cf. Encrypt-then-MAC). We assume that neither the *Luca Server* nor the *Venue Owner* can benefit from altering enc_data in any meaningful way. Instead, the verification_tag binds the *Check-In*'s timestamp to the *data secret* to avoid replay attacks by an adversary that learned about enc_data but not the *data secret*. Otherwise, said adversary might use enc_data to create *Check-In*s with the identity of the *Guest* that owns the *data secret*. Binding the timestamp to the data secret mitigates this replay to a short window of opportunity (about one minute) assuming that the *Scanner Frontend* validates that timestamps in Check-Ins are recent.

Authenticity of the HDSKP

The *HDSKP* is generated in the *Health Department Frontend* during *the registration process* and remains known exclusively to the respective *Health Department*. Currently, the *Health Department*s provide only verbatim public keys as HDSKP. A *future version of Luca* will also provide means to verify the authenticity of the HDSKP against a trusted third party.

QR Code Construction

device_type	value
iOS	0x00
Android	0x01
Static	0x02
Web App	0x03
Form	0x04

The App then displays a QR code containing:

- version (QR code protocol version)
- device_type
- key_id (ID of the *daily keypair* used for this Check-In)
- timestamp
- trace_id
- enc_data
- ephemeral_keys.public
- verification_tag
- checksum

The payload is concatenated and encoded with ASCII85 to be displayed as a QR code.

QR Code Scanning, Validation and Check-In Upload

The *Scanner Frontend* reads the above QR code using either a mobile phone camera or a dedicated scanner hardware. Before doing any further processing, it validates the `checksum` to detect reader errors. Furthermore, the contained `timestamp` is compared to the Scanner's local clock with a reasonable grace period. If either the `checksum` or the `timestamp` checks fail, no further processing is performed. Any further cryptographic validity checks cannot be performed by the *Scanner Frontend*.

Next, the relevant *Check-In* data fields are encrypted by the *Scanner Frontend* using the *venue keypair*'s public key (whose private key is in possession of the *Venue Owner*) as follows:

```
eph_scanner_keys = a new secp256r1 key pair (for DLIES with the venue public key)
dh_key           = ECDH(eph_scanner_keys.private, venue_keypair.public)
enc_key          = SHA256(dh_key || 0x01)  # truncated to 16 bytes
auth_key         = SHA256(dh_key || 0x02)
iv               = random_bytes(16)

version       = 0x03  # protocol version of the encrypted data record
check_in_data = version || key_id || ephemeral_keys.public || verification_tag ||␣
↪enc_data

venue_enc_data     = AES-128-CTR(check_in_data, enc_key, iv)
venue_enc_data_mac = HMAC-SHA256(venue_enc_data, auth_key)
```

At last, the following data is uploaded to the *Luca Server* for each successful *Check-In*.

- `trace_id`
- `scanner_id`
- `device_type`
- `timestamp`
- `venue_enc_data`
- `venue_enc_data_mac`
- `iv`
- `eph_scanner_keys.public`

Note: *luca* stores both the time sent and the time the request was received

When storing this information the *Luca Server* associates it with the `venue_id` (determined via the `scanner_id`) and the Check-In time. No further processing is done in the Luca Server.

Security Considerations

Second Layer of Encryption

As required by *O6: Venue Consent*, the *Health Department* shall be prevented from single-handedly decrypting *Guest*'s *Contact Data*. The Luca system is designed to "replace" the paper-based guest lists in physical venues that provide the same security guarantee. Hence, *Scanner Frontend*s encrypt the already encrypted *Contact Data* in *Check-In*s and remove this encryption layer only on authoritative request of a *Health Department*. See *Tracing the Check-In History of an Infected Guest* for further details.

Authenticity of venue keypair

When encrypting the (encrypted) user data (`enc_data`) and the additional data with the *venue keypair*'s public key the authenticity of that public key is crucial. Plese refer to the *security considerations regarding Venue Registration* for further details.

`scanner_id`

The `scanner_id` sent as part of the Check-In data is the only indicator *luca* can use in order to infer the associated venue. Forging a non-existent `scanner_id` could potentially allow an attacker to send bogus data to the *Luca Server*. However, this does not reveal any information to the attacker in any scenario.

On a similar note, knowing the `scanner_id` of a venue basically allows the impersonation of the venue's *Scanner Frontend*. This is accepted; more specifically, this is specifically desired in the *Self Check-In scenario*.

QR Code Scanning Feedback

The described process relies on the uni-directional communication from the *Guest App* to the *Scanner Frontend* to perform a *Check-In* by scanning a dynamic QR code. Theoretically, this allows Guest Check-Ins even without a constant internet connection of the Guest App. Nevertheless, user feedback by the *Guest App* for a successfully scanned QR code is seen as desirable.

Therefore, the *Guest App* polls the *Luca Server* via an unauthenticated connection. This inquires whether a Check-In was uploaded by a *Scanner Frontend* with a *trace ID* that the *Guest App* recently generated. Once this inquiry polling request is acknowledged by the *Luca Server*, the *Guest App* assumes that a successful QR code scan and Check-In was performed. Some UI feedback is provided to the *Guest*.

Security Considerations

This polling request might leak information about the association of a just checked-in *trace ID* and the identity of the *Guest* (directly contradicting *O2*). As mobile phone network typically use NAT, the fact that the *Luca Server* does not log any IP addresses and the connection being unauthenticated, we do accept this risk.

CHAPTER

TWELVE

CHECK-IN VIA A PRINTED QR CODE

This variation allows the *Guest App* to create *Check-In*s by scanning a printed QR code. For instance, a restaurant might place such a QR code on each available table. In contrast to the *conventional check-in* the *Scanner Frontend* is not involved. Instead, the *Guest App* assumes the role of the scanner and generates a *Check-In* single-handedly.

12.1 Overview

Participants

- *Guest*
- *Guest App*
- *Luca Server*

Assets

- *Check-In*

Preconditions

- the *Guest App* is installed
- the *Guest is registered*
- the *Venue Owner is registered*

Postconditions

- the *Guest* has created a *Check-In* at the *Venue Owner*'s venue
- the *Luca Server* has an encrypted record of the above *Check-In* event
- the *Guest App* has noted the visited location in its local *Check-In History*

12.2 Secrets

The following *secrets* are involved in this process:

Secret	Use / Purpose	Location
data secret	The *Guest*'s secret seed to derive both the *data encryption key* and the *data authentication key*. This seed will be encrypted for the *Health Department* in this process and protects the Guest's *Contact Data* stored on the *Luca Server* (cf. *contact data reference*).	Securely stored locally on the mobile device (see *Guest Registration* for further details).
tracing secret	Used to generate an anonymous *trace ID* to facilitate *contact tracing* by the *Health Department* (after the *Guest* granted access to the *tracing secret* – rendering them an *Infected Guest*).	Securely stored locally on the mobile device.
daily keypair	Used to encrypt the above-mentioned *data secret* on the *Guest*'s mobile device before transferring it to the *Scanner Frontend* via a QR code.	The public key is obtained from the *Luca Server*[1]. The private key is known to all *Health Department*s. (see *Daily Public Key Rotation* for further details).
venue keypair	Establishes a second encryption layer for the *contact data reference* that is already encrypted with the *daily keypair*.	Public key is known to the *Luca Server* and downloaded by the *Guest App* while checking in. Private key is stored by the *Venue Owner*.

12.3 Process

In this variation the *Guest App* conceptually assumes the role of the *Scanner Frontend* as described in *the conventional Check-In process*. The *Guest App* gains all required information from printed QR codes provided by the *Venue Owner*.

Printed QR Code Generation

To facilitate this feature, the *Venue Owner* generates and provides QR codes that encode the following information:

- a valid *scanner ID* for their venue
- Optional: pre-defined *additional data* fields

Those QR codes are then printed and visibly placed at the venue for *Guest*s to scan using the *Guest App*. For instance, in a restaurant the *Venue Owner* might place a unique QR code on each table and note the table number in the QR code's additional data.

Check-In via the Guest App

To check-in, *Guest*s scan the printed QR code using their *Guest App*. The *Guest App* can now use the *scanner ID* encoded in the QR code to retrieve the *venue keypair*'s public key from the *Luca Server*.

The *Guest App* now proceeds just like for *the conventional Check-In process*. Most notably, it fetches and validates the *daily keypair*[2] from the *Luca Server*, generates a valid *trace ID* using its *tracing secret* and a *contact data reference* (encrypted for the *daily keypair*).

In *the conventional Check-In process* this data would now be encoded into a QR code and presented to a *Scanner Frontend* to finish up the *Check-In* and upload it to the *Luca Server*. Instead, the *Guest App* re-encrypts the

[1] The provided public key of the *daily keypair* is signed by a *Health Department* using their *HDSKP*. This signature is provided by the *Luca Server* along with said public key.

generated data for the *venue keypair*, associates it with the *scanner ID* and uploads the finalized *Check-In* to the *Luca Server* itself.

The resulting *Check-In* is equivalent to a *Check-In* performed by the *Scanner Frontend*.

12.4 Security Considerations

Authenticity of the venue keypair

The printed QR code merely contains a *scanner ID* which is used to fetch the public key of the *venue keypair* from the *Luca Server*. At the moment, there is no way for the *Guest App* to validate the authenticity of this public key. A later version of the printed QR codes will contain the *venue keypair*'s public key directly.

Note, however, that the impact of a malicious public key is limited in this scenario as it only affects the outer layer of the *contact data reference*'s encryption. The *contact data reference* is still encrypted for the *daily keypair* and thus only accessible for the *Health Department*. Nevertheless, a theoretical collusion between the *Luca Server* and the *Health Department* could still harm security objective *O6*.

Until the mentioned improvement is implemented, this risk is accepted.

Authenticity of Printed QR Codes

By nature, QR codes are easily forgable by simply copying them. Hence, an attacker might maliciously replace QR codes of one venue with another one. This would lead to misguided *Check-In*s by *Guest*s and eventually generate false information for *Health Department*s during *contact tracing*.

As the Luca system cannot appropriately protect itself from such attacks, it relies on the *Venue Owner* to make sure that printed QR codes in their venue are not physically replaced by an attacker.

Direct Communication of Guest App and Luca Server

In contrast to the *conventional Check-In process*, the *Guest App* actively uploads its *Check-In* data to the *Luca Server*. This might allow the association of user-specific meta-data (e.g. their IP address) and the *Check-In*'s *trace ID* by the *Luca Server* (directly contradicting security objective *O2*). As mobile phone networks typically use NAT, the fact that the *Luca Server* does not log any IP addresses and the request being unauthenticated, we do accept this risk.

CHAPTER

THIRTEEN

GUEST CHECKOUT

For effective *contact tracing* the *Health Department*s must know in what time frame an *Infected Guest* was present at any given location. Hence, *Guest*s must check-out of locations when they leave.

13.1 Overview

Participants

- *Guest*
- *Guest App*
- *Venue Owner*
- *Venue Owner Frontend*
- *Luca Server*

Assets

- *Check-In*

Preconditions

- the *Guest* recently created a *Check-In* at some venue

Postconditions

- the *Guest*'s *Check-In* has a specific time period of stay

13.2 Secrets

This process requires no cryptographic secrets.

13.3 Checkout Process

Individual *Check-In*s are identified by their *trace ID* that is generated during the Check-In process (via the Guest App and *a QR code scanner*, *scanning a printed QR code* or a *static badge and QR code scanner*[1]).

For a checkout of some previous *Check-In*, the respective *trace ID* and the current timestamp are sent to the *Luca Server*. No further authentication or validation is performed and the *Check-In* is annotated with the provided timestamp.

The actual checkout might be performed in one of the following ways:

[1] Currently, there is no way for a *Guest* that uses a *Badge* instead of the *Guest App* to perform a manual checkout. See also *Inaccurate or Tampered Checkout Times*. *luca* might implement such feature in the future.

Manual App Check-out

After a *Guest* checked in using the *Guest App* they are presented with a “Check out” button for the currently active *Check-In*. Upon user request the *Guest App* informs the *Luca Server* as described above and terminates the *Check-In*. The *Guest* may now perform another *Check-In* at some other location.

Automatic Check-out via a Geofence around the Current Venue

For an automatic checkout the *Venue Owner* must provide their venue’s geo location and a “Check-In radius” (geofence) in the *Venue Information* during *initial venue registration*. Once the *Guest* physically leaves the venue’s radius, the mobile operating system will inform the *Guest App* which performs the checkout automatically.

Manual Venue Owner Check-out

*Venue Owner*s can checkout all active *Check-In*s for their venue via the *Venue Owner Frontend*. In that case, the *Venue Owner Frontend* informs the *Luca Server* about the *Venue Owner*’s wish to end active *Check-In*s at their venue. For instance, restaurants might use this to end all remaining active *Check-In*s after they close down for the day.

Time-based Check-out after 24 hours

Regardless of the checkout mechanisms described above, any *Check-In* is automatically checked out after 24 hours by the *Luca Server*.

13.4 Security Considerations

Inaccurate or Tampered Checkout Times

Checkouts must use the *trace ID* to reference their respective *Check-In* to the *Luca Server*. As the *trace ID* is *designed to be anonymous*, *luca* cannot give any authenticity guarantees regarding the stored checkout time. Any implementation trade-offs to extend *luca’s* guarantees for the checkout time would have had an influence on security objectives *O2* and *O3*.

It is worth noting that a *Health Department* usually does not blindly follow Luca’s data records when identifying likely contact persons of an *Infected Guest*, but draws educated real-world conclusions from them. Therefore, any checkout times are merely seen as a hint for real-world *contact tracing* activities by a *Health Department*.

Usage of Geo-Location Data by the Operating System

The above-described geo fence is implemented locally so the *Guest*’s location is never stored or sent to the *Luca Server*. Additionally, the *Guest* must consent to the usage of location services by the *Guest App* to use this feature. If they deny consent, they can still use *luca* but will need to always remember to checkout manually.

CHAPTER

FOURTEEN

ADDITIONAL CHECK-IN DATA

luca provides the functionality to associate a *Check-In* with additional data. This can be done by either the *Guest App* or by the *Scanner Frontend*. Additional data is designed to be non-sensitive data that can be used to narrow down possible contact persons among all guests of a venue. For instance, this might be the table number a *Guest* was placed at in a restaurant. Regardless of which app creates the *Check-In*, the additional data is encrypted using the *venue keypair* before being uploaded.

CHAPTER

FIFTEEN

GENERATION OF STATIC BADGES

A static *Badge* is a small key fob with a QR code printed on one side and a *badge serial number* on the flip side. It provides an alternative to the *smartphone based Check-In* for less tech-savvy *Guest*s. The QR code for Badges are generated by a *Trusted 3rd Party* and manufactured in bulk by a print shop. Prior to the first *Check-In Guest*s must personalize their Badge with their *Contact Data* using the *Badge Personalization Frontend*.

This chapter describes how the *Badge*s are generated. Subsequent chapters describe how they are *personalized* and *used for check-in*.

15.1 Overview

Participants

- *Trusted 3rd Party*
- *Luca Server*

Components/Assets

- *Badge*
- *Badge Generator*

Preconditions

- the *Luca Server* is equipped with a *badge keypair* and a *badge attestation keypair*
- the *Trusted 3rd Party* is ready to use the *Badge Generator*
  - possesses the required Luca API token[1]
  - received the public key of a valid *badge keypair*

Postconditions

- one or more new *Badge*s are generated and registered in the *Luca Server*

15.2 Secrets

The following *secrets* are involved in this process:

[1] For specific tasks the Luca API requires special authentication via a so-called "Bearer token". Further details about the authentication mechanism of the *Luca Server* are beyond the scope of this chapter.

Secret	Use / Purpose	Location
badge serial number	A random seed to derive the *Badge*'s intrinsic secrets required to both encrypt and associate *Contact Data* and perform *Check-In*s	Defined by the *Badge Generator* and printed onto the flip side of the *Badge*. Entered into the *Badge Personalization Frontend* to associate *Contact Data* to a *Badge*.
data secret, *data encryption key*, *data authentication key*, *tracing secret*	Those secrets are required for the encryption of *Contact Data* and performing *Check-In*s at *luca* locations. They are derived from the before-mentioned *badge serial number*.	Transiently known to the *Badge Generator*. Later re-derived by the *Badge Personalization Frontend* for encrypting/associating *Contact Data*.
`guest keypair`	Used to sign the encrypted *contact data reference* on the badge and (during *Badge Personalization*) to authenticate the owner of the Badge. This keypair is the Badge-equivalent of the *guest keypair* held by the *Guest App*.	The private key is transiently known to the *Badge Generator* as it is derived from the *badge serial number*. Later it is re-derived by the *Badge Personalization Frontend* for authenticating the *Contact Data*. The public key is stored on the *Luca Server*.
badge keypair	Used to encrypt the *contact data reference* while generating a *Badge*.	The public key is obtained from the *Luca Server*[2]. The private key is known to all *Health Departments*.
badge attestation keypair	Used to sign freshly registered *Badges* for later validation by the *Scanner Frontend*.	The private key is kept in the *Luca Server* for a below-described remote attestation workflow. The *Scanner Frontend* can access the associated public key for validation.

15.3 Process

The *Badge Generator* software creates each *Badge* by randomly choosing a 56-bit serial number. All cryptographic assets and the associated *user ID* for the generated *Badge* are then derived from this initial seed (aka. the *badge serial number*). The newly generated *Badge* is then registered with the *Luca Server* via an authenticated API request[7]. In response, the *Luca Server* creates a remote attestation signature for the badge using its *badge attestation keypair*. Eventually, the relevant information is encoded into a QR code and printed on a plastic key fob along with the initially generated serial number.

The *Badge Generator* runs the following algorithm to generate a *Badge*:

```
# pseudocode

# generate random bytes and use argon2id to derive initial keying material
entropy = random_bytes(7)
seed    = argon2id(entropy, salt="da3ae5ecd280924e",
                   length=16, memorySize=32MiB, iterations=11, parallelism=1)

# generate key material that should be available via the Badge's serial number only
level_one     = HKDF-HMAC-SHA256(seed, length=64,
                                 context="badge_crypto_assets",
                                 salt="")
data_secret   = level_one[0:16]
tracing_seed  = level_one[16:32]
guest_keypair = level_one[32:64]

# generate additional key material that needs to be available to a Scanner␣
↪Frontend scanning the Badge
level_two              = HKDF-HMAC-SHA256(tracing_seed, length=48,
```

(continues on next page)

[2] The provided public key of the *badge keypair* is signed by a *Health Department* using their *HDSKP*. This signature is provided by the *Luca Server* along with said public key.

(continued from previous page)

```
                                          context="badge_tracing_assets",
                                          salt="")
user_id                 = toUuid4(level_two[0:16])
badge_verification_key = level_two[16:32]
tracing_secret          = level_two[32:48]

# encrypt a 'blanco' Contact Data Reference using the badge keypair's public key
iv                    = guest_keypair.public # public key bits truncated to 16 bytes
ephemeral_keys        = newSecp256r1Keypair()  # for DLIES with the badge keypair␣
↪public key
dh_key                = ECDH(ephemeral_keys.private, badge_keypair.public)
enc_key               = HMAC-SHA256(data=dhKey, key=iv) # truncated to 16 bytes
enc_contact_data_ref = AES-128-CTR(user_id || data_secret, enc_key, iv)

# sign the public Badge data with the guest_keypair
signature = sign(guest_keypair.private, user_id || enc_contact_data_ref)

# register the Badge with the Luca Server and receive an
# attestation of the new Badge using the Badge Attestation Key in response
attestation_signature = httpPOST("/api/v3/users/badge",
                                 {
                                    "userId":    user_id,
                                    "publicKey": guest_keypair.public,
                                    "data":      enc_contact_data_ref
                                    "signature": signature
                                 })
```

QR code and Serial Number Contents

The *Badge*'s QR code is then constructed by concatenating the following data:

- badge revision (currently `0x04`)
- device type (`0x02` for "Badge")
- *badge keypair* ID (1 byte identifier of the *badge keypair* used to encrypt the *contact data reference*)
- `tracing_seed`
- `enc_contact_data_ref`
- `attestation_signature` (in IEEE 1363 format)
- `badge_ephemeral_public_key` (`ephemeral_keys.public` from above)
- checksum (SHA-256 of all the previous data truncated to 4 bytes)

The created data buffer is then encoded using the Z85 encoding and printed onto a plastic key fob in the form of a QR code. Along with the QR code each *Badge* features its associated *badge serial number* that is the random `entropy` value encoded in base32crockford and split into four digit groups (e.g. `LUCA-1337-COOL`).

Rationale of Generated and Exposed Secrets

The Badge secret derivation is split into three steps:

1. *Hardening against brute force attacks on the short 56-bit serial number.* Ensured by employing the password hash algorithm `argon2id`. See *the security considerations* for a discussion on the chosen tuning parameters.
2. *Derivation of 'secret' values that only the owner of the Badge should have access to.* The derive those values one needs to know the *badge serial number* with the exception of the `tracing_seed` which is exposed in plain in the Badge's QR code. Secret values are required to *personalize the Badge* or perform contact tracing with it.
3. *Derivation of 'public' values that are exposed in the QR code.* These values are revealed in the QR code and allow the *Scanner Frontend* to perform a *Check-In* in the name of the Badge's owner.

All public values revealed on the printed key fob are discussed below.

"badge revision" and "device type"

Those are technical values needed by the *Scanner Frontend* to distinguish a *Badge* from a *Guest App* (device type) and allow for schema updates (badge revision) of the described Badge process.

badge serial number

The Badge's owner can use the badge serial number printed on their key fob for two things:

1. Personalize the Badge by associating their *Contact Data* with the Badge. For further details see *Badge Personalization*.
2. Facilitate a *contact tracing* by revealing the Badge's serial number to the *Health Department*

In both cases, either the *Badge Personalization Frontend* or the *Health Department Frontend* will re-derive all necessary secrets from the badge serial number for the desired use case.

badge keypair ID and `attestation_signature`

The *badge keypair* ID identifies the badge keypair that the *Luca Server* used to sign the badge data during the registration. Using the public key of the matching *badge keypair* the *Scanner Frontend* can check the `attestation_signature` of the scanned *Badge* and determine whether it is valid or not. See *the badge Check-In process* for further details.

`tracing_seed` and Encrypted contact data reference (`enc_contact_data_ref`)

Using the `tracing_seed` the *Scanner Frontend* can derive the badge owner's *user ID* and *tracing secret*. With this information, the *Scanner Frontend* is able to create a valid *Check-In* for the *Badge*'s owner. See *the badge Check-In process* for further details.

15.4 Security Considerations

Choice of `argon2id` Parameters

The parameters for the `argon2id` key derivation function are optimized to maximally slow down a brute-force attack, while not prohibitively inconveniencing users of low-end commodity hardware in the *Badge Personalization flow*. The specific parameters (`memorySize`: 32MiB, `iterations`: 11, `parallelism`: 1) yield an execution time of around 0.5 seconds using a native implementation of the on high-end consumer hardware and around 2 to 5 seconds using a browser-based JavaScript/WebAssembly implementation on a mid-range to low-end mobile device.

A random but fixed salt is used. This is not a problem, as `argon2id` is not used for password hashing here but for key derivation from random input data (termed `entropy` above).

CHAPTER

SIXTEEN

BADGE PERSONALIZATION

After the *Badge* has been created *as described above*, it contains an encrypted contact data reference (enc_contact_data_ref). This reference is conceptually very similar to the *contact data reference* used by the *Guest App*. However, at this point there is no contact data associated to the reference, yet. Guests need to personalize their *Badge* using the *Badge Personalization Frontend*.

16.1 Overview

Participants

- *Guest*
- *Luca Server*

Components/Assets

- *Badge*
- *Contact Data*

Preconditions

- there is a *Guest* that is willing to use a *Badge* for *Check-In*s
- the *Guest* has received their *Badge*
- the *Badge*s was generated as described in *Generation of Static Badges*

Postconditions

- some *Guest* has personalized their *Badge* with their *Contact Data* via the *Badge Personalization Frontend*

16.2 Process

The *Badge Personalization Frontend* requires the *badge serial number* and the Guest's *Contact Data*. It creates the *encrypted guest data* as follows:

```
# pseudocode

# derive the initial keying material from the serial number
seed = argon2id(entropy, salt="da3ae5ecd280924e",
                length=16, memorySize=32MiB, iterations=11, parallelism=1)

# derive secrets analogously to the Badge Generation process
level_one     = HKDF-HMAC-SHA256(seed, length=64,
                                 context="badge_crypto_assets",
                                 salt="")
data_secret   = level_one[0:16]
tracing_seed  = level_one[16:32]
```

(continues on next page)

(continued from previous page)

```
guest_keypair = level_one[32:64]

level_two = HKDF-HMAC-SHA256(tracing_seed, length=48,
                             context="badge_tracing_assets",
                             salt="")

user_id                = toUuid4(level_two[0:16])
badge_verification_key = level_two[16:32]
# tracing_secret is not required in this process

# derive the symmetric encryption key from the data secret
# this key directly corresponds to the data_encryption_key used by the Guest App
data_encryption_key = SHA256(data secret || 0x01) # truncated to 16 bytes

# encrypt contact_data and badge_verification_key analogously to how the Guest
# App creates the encrypted_guest_data
# the badge_verification_key corresponds to the data_authentication_key

iv = random_bytes(16)

encrypted_guest_data = AES_128(contact_data + badge_verification_key,
                               key=data_encryption_key,
                               mode=CTR,
                               iv=iv)

badge_data_mac = HMAC(encrypted_guest_data,
                      key=badge_verification_key)

badge_data_signature = guest_keypair.private.sign(encrypted_guest_data +
                                                  badge_verification_key +
                                                  iv)
```

The *Badge Personalization Frontend* sends the following data to the *Luca Server*:

- the *encrypted guest data*
- the `IV` used in the encryption
- the `badge data mac`
- the `badge data signature`

The *Luca Server* verifies that the request is authorized by checking the provided signature with public key that was uploaded when the *Badge was generated*.

CHAPTER

SEVENTEEN

BADGE CHECK-IN

In order to check-in with a Static *Badge* the Guest presents the Badge's QR code to the *Scanner Frontend*, the same way they would if they were using the *Guest App*. However, as the static QR code on the Badge cannot dynamically create *trace IDs*, the *Scanner Frontend* has to assume some of the tasks normally done by the Guest App.

The Scanner Frontend reads following information from the QR code

- `enc_contact_data_ref`
- `badge_keypair_ID`
- `tracing_seed`
- `badge_ephemeral_public_key`
- `attestation_signature`

and uses it to create a *Check-In* as follows:

```
# pseudocode

# the tracing_seed was transmitted via the scanned QR code
level_two              = HKDF-HMAC-SHA256(tracing_seed, length=48,
                                 context="badge_tracing_assets",
                                 salt="")
user_id                = toUuid4(level_two[0:16])
badge_verification_key = level_two[16:32]
tracing_secret         = level_two[32:48]

# the data created below corresponds directly to the data in the QR code displayed␣
↪by the Guest App
timestamp        = UNIX timestamp rounded down to the last full minute (little␣
↪endian encoding)
trace_id         = HMAC-SHA256(user_id || timestamp, tracing_secret)  # truncated␣
↪to 16 bytes

enc_data         = enc_contact_data_ref  # printed on the QR code
verification_tag = HMAC-SHA256(timestamp || enc_data, badge_verification_key)
```

The rest of the check-in procedure is equivalent to the *Check-In via the Mobile Scanner App*.

17.1 Security Considerations

A Static *Badge* cannot provide the same security guarantees as the *Guest App*. During check-in, the *Scanner Frontend* learns the Badge's *tracing secret* and performs tasks that would normally be done by the Guest App.

As the QR code printed on the Badge is immutable and is the only asset required in order to check-in using the Static Badge, the Scanner Frontend now possesses all knowledge required to perform a check-in in the name of the Guest. Obviously, it also allows the Scanner Frontend to recognize Badges it had previously scanned.

CHAPTER

EIGHTEEN

TRACING THE CHECK-IN HISTORY OF AN INFECTED GUEST

The goal of this process is to identify *Guest*s that are at a risk of being infected, as a result of having been in contact with an *Infected Guest*. The process consists of two major parts: *Tracing the Check-In History of an Infected Guest* and *Finding Potential Contact Persons*. This chapter describes the first part.

18.1 Overview

Participants

- *Infected Guest*
- *Guest App*
- *Health Department*
- *Health Department Frontend*
- *Luca Server*

Assets

- *Contact Data*
- *Check-In*
- *Check-In History*

Preconditions

- the *Infected Guest is registered*
- the *Infected Guest* has created at least one *Check-In* (see *Check-In via Mobile Phone App*)[1]
- the *Infected Guest*'s *Guest App* has retrieved and validated the public key of the current *daily keypair*
- the *Health Department* is onboarded to the Luca system and has access to the *daily keypair*s (see *Daily Keypair Rotation*)
- the *Health Department* is in contact with an *Infected Guest*

Postconditions

- the *Health Department* has access to the *Check-In History* of the *Infected Guest*

[1] The process can also be trivially performed if the Guest has not created any *Check-In*s but the *Check-In History* will be empty.

18.2 Secrets

The following *secrets* are involved in this process:

Secret	Use / Purpose	Location
*tracing secret*s	Given the consent of the *Infected Guest* the relevant *tracing secret*s are made available to the *Health Department* and the *Luca Server* to reconstruct the *Check-In History*.	• *Guest App* • *Health Department Frontend* • *Luca Server*
data secret	During the process, the *Health Department Frontend* fetches the *Infected Guest*'s *encrypted guest data*, decrypts it using the *data secret*, and displays it.	• *Guest App* • *Health Department Frontend*
daily keypair	The *Guest App* encrypts the *guest data transfer object* with the public key. The *Health Department* uses the private key for decryption of the same.	• *Health Department Frontend* • *Guest App* (public key only)
tracing TAN	The TAN is created on the *Luca Server* as an identifier for the encrypted *guest data transfer object* by request of the *Guest App*, which displays it to the Guest. The Guest then communicates it to the *Health Department*.	• *Luca Server* • *Guest App* • *Health Department Frontend*

18.3 Process

```
<IPython.core.display.SVG object>
```

The first part of the contact tracing is for the *Health Department* to reconstruct the *Check-In History* of the *Infected Guest*. Each *Check-In* stored in *luca* is associated with an unique *trace ID*. These IDs are derived from the *tracing secret* stored in the *Guest App* (as well as from the Guest's *user ID* and a timestamp). Hence, given the *Infected Guest*'s *tracing secret*s the *Health Department* can reconstruct the *Infected Guest*'s *trace ID*s and find all relevant *Check-In*s.

Accessing the Infected Guest's Tracing Secrets

In the first step the *Health Department* needs to acquire the *Infected Guest*'s *tracing secret*s for the epidemiologically relevant timespan. Each *tracing secret* will allow the *Health Department* to find all *Check-In*s whose *trace ID* is based on this secret.

In the beginning of the process, an *Infected Guest* is directly contacted by a local *Health Department*. In order to retrieve the Guest's *tracing secret*s the Health Department asks the Guest to reveal their *Contact Data* and *Check-In History* via a functionality in the *Guest App*.

When this functionality is activated, the App creates a *guest data transfer object* that holds all information required for the Health Department's tracing process:

Asset	Use
*tracing secret*s	Needed to reconstruct the Guest's *trace ID*s
user ID	Needed to reconstruct the Guest's *trace ID*s
data secret	Used to validate and display the *Infected Guest*'s identity in the *Health Department Frontend*

The data is encrypted using the current *daily keypair*'s public key[2] and uploaded to the *Luca Server*. The Luca Server returns a *tracing TAN*, which is a short alpha-numeric identifier for the *guest data transfer object* on the Luca Server and does not carry any further security relevance.

The *Infected Guest* can now pick up their communication with the *Health Department* and spell out the *tracing TAN*. This allows the Health Department to retrieve the encrypted *guest data transfer object* from the Luca Server. The transfer object is decrypted using the *daily keypair*'s private key. Usage of the *daily keypair* within the *Health Department* is detailed in Chapter *Daily Keypair Rotation*.

After a short timespan of a few hours the encrypted *guest data transfer object*s are deleted from the *Luca Server*.

Reconstructing the Infected Guest's Check-In History

The second step is for the *Health Department* to find all venues where the *Infected Guest* has created *Check-In*s within the recent, epidemiologically relevant timespan (e.g. 14 days).

To start the tracing process, the *Health Department* sends the *Infected Guest*'s *tracing secret*s to the *Luca Server*. Based on the secrets and the affected *user ID*, the Luca Server generates all possible *trace ID*s for the relevant time frame. Given these *trace ID*s *luca* can find all *Check-In*s created by that Guest during that time frame—the *Infected Guest*'s *Check-In History*.

The *Luca Server* can use the recovered *Check-In History* to contact all venues the *Infected Guest* has visited. The process of contacting *Venue Owner*s for lifting the outer layer of encryption in each affected *Check-In* is described in the *next part*.

18.4 Security Considerations

Correlation of Guest Data Transfer Objects and Encrypted Guest Data

After receiving a *Infected Guest*'s *guest data transfer object* the *Health Department Frontend* uses the contained *user ID* to obtain that Guest's *encrypted guest data* from the *Luca Server*. This is done in order to display the *Infected Guest*'s *Contact Data* to the *Health Department*.

The *Luca Server* can (indirectly) use this circumstance in order to associate a *guest data transfer object* with the *encrypted guest data* of the same Guest by observing the *Health Department Frontend*'s requests.

Accidental Upload of Guest Data Transfer Object

Guests could trigger the *Guest App*'s functionality to upload the *guest data transfer object* and request a TAN accidentally or out of curiosity. This would needlessly upload their encrypted secrets, but they still would not be accessible to the *Luca Server* (as they are encrypted for the *daily keypair*) nor the *Health Department* (as they do not know the TAN).

We believe this risk is acceptable and can further be mitigated by an informative warning message in the *Guest App* when activating the functionality.

---

[2] Whenever making use of the *daily keypair* the *Guest App* verifies the key's validity and authenticity as described in *Daily Keypair Rotation*.

CHAPTER

NINETEEN

FINDING POTENTIAL CONTACT PERSONS

19.1 Overview

Participants

- *Traced Guest*s
- *Luca Server*
- *Health Department*
- *Health Department Frontend*
- *Venue Owner*
- *Venue Owner Frontend*

Assets

- *Contact Data*
- *Check-In*
- *Check-In History* (acquired in the *previous part*)

Preconditions

- the *Health Department* has the *Check-In History* of the *Infected Guest*

Postconditions

- the *Health Department* has access to the *Contact Data* of all *Traced Guest*s

19.2 Secrets

The following *secrets* are involved in this process:

Secret	Use / Purpose	Location
data secret	The *data secret*s of all *Traced Guest*s are made accessible to the *Health Department* in the process in order to decrypt the *encrypted guest data*.	• doubly encrypted in each *Check-In* • encrypted with the *daily keypair*, then decrypted, in the *Health Department Frontend*
venue keypair	The keypair's private key is used in the *Venue Owner Frontend* to decrypt the outer layer of encryption on the *contact data reference* and the Additional Data in each *Traced Guest*'s *Check-In*.	• *Venue Owner Frontend*
daily keypair	The keypair's private key is used in the *Health Department Frontend* to decrypt the inner layer of encryption on the *contact data reference* in each *Traced Guest*'s *Check-In*.	• *Health Department Frontend*

19.3 Process

```
<IPython.core.display.SVG object>
```

Given the *Infected Guest*'s *Check-In History* (obtained in *part 1 above* the *Luca Server* determines all *Venue Owner*s whose venues have been visited by this Guest. Each of them is contacted using the *Venue Information* provided during *Venue Registration* and asked to reveal the encrypted *contact data reference*s of potential contact persons (*Traced Guest*s).

When a *Venue Owner* has been contacted to assist in contact tracing they use the respective functionality in the *Venue Owner Frontend*. The *Venue Owner Frontend* proceeds to download all *Check-In*s registered for this venue that coincide with the visit of the *Infected Guest* from the *Luca Server*. For each of these *Check-In*s both the outer encryption layer on the *contact data reference* and the Additional Data are decrypted using the private key of this venue's *venue keypair*. Note that, after the decryption, the *contact data reference* is still encrypted with they *daily keypair* and thus only accessible to the *Health Department* (see *Check-In via Mobile Phone App*).

The data is uploaded back to the *Luca Server* in order to be shared with the *Health Department* that initiated the tracing.

After decrypting the *contact data reference*s the Health Department possesses each *Traced Guest*'s *user ID* and *data secret*. It fetches the Guests' *encrypted guest data* from the *Luca Server* using the *user ID* and decrypts it using the *data encryption key* (derived from the *data secret*) to obtain the *Contact Data* and the *data authentication key*. The latter is used to verify the authenticity of both the *Check-In* and the *contact data reference*. The *Contact Data* can now be used to contact the *Traced Guest* and inform them that they are at risk of being infected.

19.4 Security Considerations

Authentication of Contact Data and Check-Ins

The *data authentication key* is used to authenticate both the *contact data reference* in the *Check-In* (using the *verification tag*) and the *Contact Data*. However, the *data authentication key* has to be retrieved by deriving the *data secret* contained in the *contact data reference*.

This is unusual (cf. Encrypt-then-MAC), but we consider it sound. Please refer to *the Security Considerations regarding Guest Check-In* for further details.

Possible Abuse of Traced Guests’ Data Secrets

In the process described above the *Health Department* obtains each *Traced Guest*’s *data secret* and derives the symmetric *data authentication key* from it. It uses this key to validate the authenticity of the *Check-In*s, verifying that the *Check-In*s it received from the *Luca Server* have in fact been created by the owner of the *data secret* (the *Traced Guest*).

However, having learned the *data secret*, the *Health Department* can now itself create apparently valid *Check-In*s for that *Traced Guest*. Neither the *Luca Server* nor another *Health Department* can distinguish these forged *Check-In*s from authentic ones.

Note that the *Health Department* does not know *Traced Guest*s’ *tracing secret*s here. Hence, the forged *Check-In*s would not appear in the Guest’s *Check-In History*. They would, however, appear whenever the forged *Check-In* coincides in time and place with another *Traced Guest*’s *Check-In*.

CHAPTER

TWENTY

CRYPTOGRAPHIC ALGORITHMS

This chapter privides details about the cryptographic algorithms used throughout *luca*. All primitives have been selected in accordance with the Technical Guideline TR-02102-1 of the German BSI[1].

20.1 Symmetric Encryption

In a future version of *luca* we plan on using `AES-128` in Galois/Counter Mode (`GCM`) instead.

luca uses `AES` with a key length of 128 bits in Counter Mode (`AES-128-CTR`) for symmetric encryption. Data is authenticated using `HMAC-SHA256`.

20.2 Asymmetric Cryptography

luca uses elliptic curves for the required asymmetric cryptography.

Encryption

Asymmetric encryption in *luca* is based on the DLIES encryption scheme as specified in section 3.5 of BSI TR-02102-1[?]. For the components required, *luca* uses the following primitives:

Component	Algorithm
asymmetric key parameters	`secp256r1` (`NIST P-256`)
symmetric encryption	`AES-128-CTR`
message authentication code	`HMAC-SHA256`
key derivation function	`SHA256`

[1] BSI TR-02102-1. Cryptographic Mechanisms: Recommendations and Key Lengths, accessed 2021/03/04

Signing

We use ECDSA with the `secp256r1` (`NIST P-256`) elliptic curve.

On the Use of `secp256r1`

The curve `secp256r1` is recommended by NIST[2] for use with Discrete Logarithm-Based Cryptography. It is, however, criticized for using unexplained inputs in the curve-generation process and hence rumored to be backdoored by the NSA[3]. At the time of writing, those rumors can neither be proven nor disproven. Consequently it would be preferable to use a different curve that is generally considered safe.

However, as of 2021, `secp256r1` is the only curve widely supported by many mobile phone hardware-based key managers[4][5]. We chose to use `secp256r1` because we value the security benefit of using the trusted module higher than the risks laid out above.

20.3 Encryption Scheme

Based on the scheme specified above, asymmetric encryption is implemented by the components in *luca* as follows:

```
# pseudocode

# given the inputs:
# * data: the data to encrypt
# * receiver_public_key: the public key of the receiver
# * iv (optional): the initialization vector

ephemeral_keys     = a new secp256r1 key pair (for DLIES with the receiver's␣
↪public key)
iv                 = random_bytes(16)  # Note: in some contexts an external input␣
↪(usually
                                       #       an ephemeral public key) is used as␣
↪IV
dh_key             = ECDH(ephemeral_keys.private, receiver_public_key)
encryption_key     = SHA256(dh_key || 0x01)  # truncated to 16 bytes
authentication_key = SHA256(dh_key || 0x02)
encrypted_data     = AES-128-CTR(data, encryption_key, iv)
mac                = HMAC-SHA256(encrypted_data, authentication_key)
```

The function's output includes `encrypted_data`, `mac`, `iv` and `ephemeral_keys.public`.

On the Use of the ephemeral public key as IV

As indicated in the pseudocode snippet above, we sometimes re-use the DLIES ephemeral key pair's public key as the initialization vector for `AES-CTR`.

Our motivation for this is the limited capacity of the Check-In QR codes generated by the *Guest App*. The receiver of the encrypted data needs both, the ephemeral public key and the IV, to decrypt the data. Re-using the public key as IV saves space.

This construction is uncommon, but secure. `AES-CTR` only requires the initialization vector to be unique, which is satisfied here.

[2] NIST SP 800-186 (Draft). Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters accessed 2021/03/04

[3] Daniel J. Bernstein and Tanja Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography., accessed 2021/03/04

[4] Apple Developer documentation on SecureEnclave, accessed 2021/03/04

[5] Android Developer documentation on HSM, accessed 2021/03/08

Decryption

Accordingly, decryption and authentication is implemented as follows:

```
# pseudocode

# given the inputs:
# * encrypted_data: the data to decrypt
# * receiver_private_key: the private key of the receiver
# * iv: the initialization vector
# * ephemeral_public_key: the sender's public key for DLIES (ephemeral_keys.public)
# * mac: the message authentication code

dh_key             = ECDH(receiver_private_key, ephemeral_public_key)
encryption_key     = SHA256(dh_key || 0x01)  # truncated to 16 bytes
authentication_key = SHA256(dh_key || 0x02)

verify_mac(mac, HMAC-SHA256(encryptedData, authentication_key))
decrypted_data     = AES-128-CTR(data, encryption_key, iv)
```

If the provided `mac` is valid, the function returns the `decrypted_data`.

CHAPTER TWENTYONE

PLANNED IMPROVEMENTS

luca is under continuous development. This appendix collects improvements we are currently working on.

21.1 Certification of Health Department Keypairs

The two keypairs of each *Health Department*, the Health Department Signing Keypair *HDSKP* and the Health Department Encryption Keypair *HDEKP*, play a crucial role in the *luca* system. As described in the chapter *Daily Keypair Rotation* the *HDSKP* is used to sign the *daily keypair*. This signature is verified by the *Guest App* during *check-in*. The *HDEKP* also plays an important role in the process of *Daily Public Key Rotation*: the new *daily keypair* is made accessible to all *Health Department*'s *HDEKP*s registered in *luca*. Consequently, the authenticity of these two keypairs is of great importance.

During the pilot phase *luca* did not rely on a third party PKI to ensure the authenticity of those public keys. Hence, *Guest App*s and *Health Department*s currently need to rely on the *Luca Server* and *Luca Service Operator* to ensure the authenticity of the keypairs.

With the country-wide rollout of *luca* the *HDSKP* and *HDEKP* public keys will be signed by certificates issued by Bundesdruckerei's subsidiary company D-Trust GmbH. Those certificates will be based on one of D-Trust's global root certificates, provide revocation mechanisms and will be issued only after manual validation of the requesting Health Department's identity. Both the *Guest App* and the *Health Department*s will then rely on a chain of trust to ensure the authenticity of *HDSKP* and *HDEKP*.

We will update this document accordingly.

CHAPTER

TWENTYTWO

CONTACT *LUCA*

22.1 GitHub

Feel free to start and contribute to discussions on the “Discussions” page on GitHub.

22.2 Security Related Issues

If you find any security issues, please get in contact via security@luca-app.de and encrypt your message with the following PGP key:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGBFyPUBEAC6f5tGTsDEfCpCAcufybw4GgB3+UoJKubFe8QOxXo8RiMvnjMO
gdQtYj4TTFpe8xvXCkAZgXzRXNn+ylF7PtM4OSIi5V+PIUfmJkR+9uQMK2FgJ6+p
zGQbem80nsGlGcLfXQGLbEvzMGBc6zcyXB3iAyIMIju4BoW5XykMO13pITorww4S
6tAz6X0oUZFwUDBAXB35fqPBVdk2D4ZZMZiPX45KFvg3gBRO66PxYG595z2RzhRj
/rY5ojN8AqACCy0aNzcTj64hdzXRSxfgeUOt9Ve59R54oVM14RM0M1w6ub8Y7uOd
couh+RAqTVs1qW8naF8ioHI3BxYRKCV93FqWvF1foOt9aNTlCo9YZWyPocd2luUm
n8nUfi7rfP4iMEKpGTa6oJ88QhrSzKtODosmo1waqNCnF3axZLvqVJ/rFy4CZAnz
0b0C4T4Q11I6vDEv7B7GKNeq/39JPREaucarjCsnyVdJM/0Whtw1bd+0sy13HFU4
X+QxcSleT/tFGvA9uw/WTfwhE+RqSaHk7YI1zO+VpPmzyQSmMUhs1pPdiQaalOIm
PzsDZIA9ifpgDR5pXbgAMKMOw6qE8UeSSyfchZSSqvxFBCTEKwSRo2ceJRdqKm9O
cBnNWDNB7ZhWJvdxT5wI4kHt3J0VcftddejFBNdCPZDEz/EFNe0EpJuJwwARAQAB
tChjdWx0dXJlNGxpZmUgR21iSCA8c2VjdXJpdHlAbHVjYS1hcHAuZGU+iQJUBBMB
CAA+FiEESCS9XGyNJ1YasIy1Ax3WxGr9CEsFAmBFyPUCGwMFCQPCZwAFCwkIBwIG
FQoJCAsCBBYCAwECHgECF4AACgkQAx3WxGr9CEv0Fg//SydWxFcN5mmVGRfwro4J
sv8imYfFpTcqTlejSTBmH/MZJwy5qU8N4W3EnvDGfEElBLYFMbmFX3gEx6Yaz28w
4BlWGEdzhMyQMTWjoRsIl/EdpyLrP8rtVXxXRLWhzGLu3x917+wQMOGrgVHYYeld
3UaSs3ioPOBZGcFT/Y2ow5alUJs7iZe+aaCIrETJkjwltQWTnOzPGGxdNHsGjZKM
+6Mg7mSpso64JyZTAWxGqyXZ645IYlIjzBDVjpkPZatooabcUZ/5KvVbbUm70W8W
DmzX170YbiwPc6eQKDB9cZJsXUjmSHaBC4NzVzFOXeELyFkn3okNwmIvfXDmKhQy
Pb6H+8dflQ1orsE7cjP5JdisPLz3LmTDq1tt/To9bQ8yntpcH+naZFvSC8rs+ruj
xw6t1lKI2L44OJ9uoOfKXxp50XYuNRwy7W+vY6EOgQwMwGZnazR6DplcHPjKfq+o
catTtRbrY/a3PjX0cKJslbaDRPa5IoTHnMskfZPKF8ckSa1SdVG1xuDQ/oD2VMAI
6+/k9mZOVt27Re4NArCkwmVwqCA6PGBdMHoL4fl0jUJx3YT3WyeawIwXgG2eTHSo
jTyjwiL4iSEJBdMD4oo+cs87EdCaeClGb1CAj5YQYNVUSyyUSV2pJz7IVllqpEU8
aPyT3cGK6XdFrNnJuu62d0e5Ag0EYEXI9QEQANdu7jnaUhogRE31xd7af1uGS6Pl
nCMpnKlWmiDuwFMF06P2eHXOdilNw5/fbYPomSaa639irg6ZZ2ayCU2fivhN7ohB
g4OAv36LVOqgtBBafuQL/1k9i8cNMpK2yrGMzh+Gwt3eZKMkMSh3YkRAVUzlzDNV
4KIwDqWQ0QsECn45yaQ4wHINpbFynZAoQ9GVYz8E3tAX76rEKyAV/tX77eRMwfbG
bYFmHXZRIk8k/4FG00ns1U943in4YWzWaKv8i4+Ip3vxa/wGf7gvD0DIx7ZHELDW
+JZsfvWLsbcoqX4aMNaXiWPi1wqV2WXnJfXQf4ReOpAx1eTmYtPN/DeLXdAFsMBx
B9O9vTo5ATpt/JMwZ6FIluQ99OMssYCb6IePQPbdgClNEEDEcehGynTOL8VHh9lA
YhFNBSGN4PxUSyvnx75d44D1GSGyD+3GjfOqiE2GvBIXQODV1gLTXivySm7wXKE0
MfffcnZOg9scWIreZ1tVjaYJ7ywv80nGK3L9/C0j83kzKJXx+neXD8k3Bx3w1pz0
GqDYVcRBmd5CdchgqnGgjQJ7tDDjGGDxOhdONqWRBHkGSU/SsgroZVtTzLQm25is
```

(continues on next page)

(continued from previous page)

```
bPdocGnrsq4RPr1abwYZgX0Dh6SuuNWDf209qDjVQJuoh0IMiprIXKNdrS7tYmxm
kknkPYFcUlLbESlFABEBAAGJAjwEGAEIACYWIQRIJL1cbI0nVhqwjLUDHdbEav0I
SwUCYEXI9QIbDAUJA8JnAAAKCRADHdbEav0IS0zqD/91KnBmJd/cfdLPnYREtfr6
tDFMSUy04OqeeyMnRiYRyr/TQnEEM/TkJfgia7LOkSY8ePxtwd9HsM7l/dsQgwP1
EABzqP7DySrqlGQlIzXd390V1A0UzSKlB2+5NGYNv8D5xtP53gvTmb2gB2V5DZD+
8E3pF+XQl9VNEUkyGcTFLLFuWWyqb8kpiGKpZKRMqIIdHbUEnseSErblVHXvnN0p
BQh4FZbFyJn+NVz6OYewZ8ytI4YqxkC9aEIjMxhCyYXFF0kWjQNx0S99fWvgdG+/
RTlv+r/G5g54S++0xLE1/5aiBsKlZEhOrN5o670h6cwefYtW+NvgEOWbT/i0maeK
3UUqYwEWztWyKA0Q+ENocQNkzThN+L6Il7YuAH0qGgqJ0sjnmmg7+gg7CqAnyO1s
loVkYWOaqXgoS0oO6gmlLAF1jDUVLgSXDAKGHyOQfE5XUjHZwruYVtH0Klo2sUQJ
xWP+YhliFj0rIXyfueT7TFVJTqxagK9RmRoKWtxzFmqbfJXIu7C7OU9EjnR3tt6u
k1bhDIgOgx4TjkNnNfH8FEGVYD7Mb1pMTcPUpS233ik87QY2/8Qod2FLpwwCtpy8
o+h/0l3DB0zmZNU8Ci0zD/PPa0ivfuoxtuUybfNaHXj+3rG9yvtO1ZZbweoaEdSL
HTmQN7E9lsxEBmyIOYoCUA==
=J3BG
-----END PGP PUBLIC KEY BLOCK-----
```

INDEX