das-FaceQR - Biometric Credential Service

Release 2021Q2 (v2.5.4)

API Specification v2

Release	Date	Description	Author	Reviewer	Approver
1.4	20/6/2021	das-Face QR release 2021Q2 (v2.5.4)	ISL	MSY	GSS

1. Introduction

The objective of this document is to introduce the das-FaceQR service, focusing on its characteristics, usage modes and its relation with other Veridas products and applications.

The process described herein is protected by a patent.

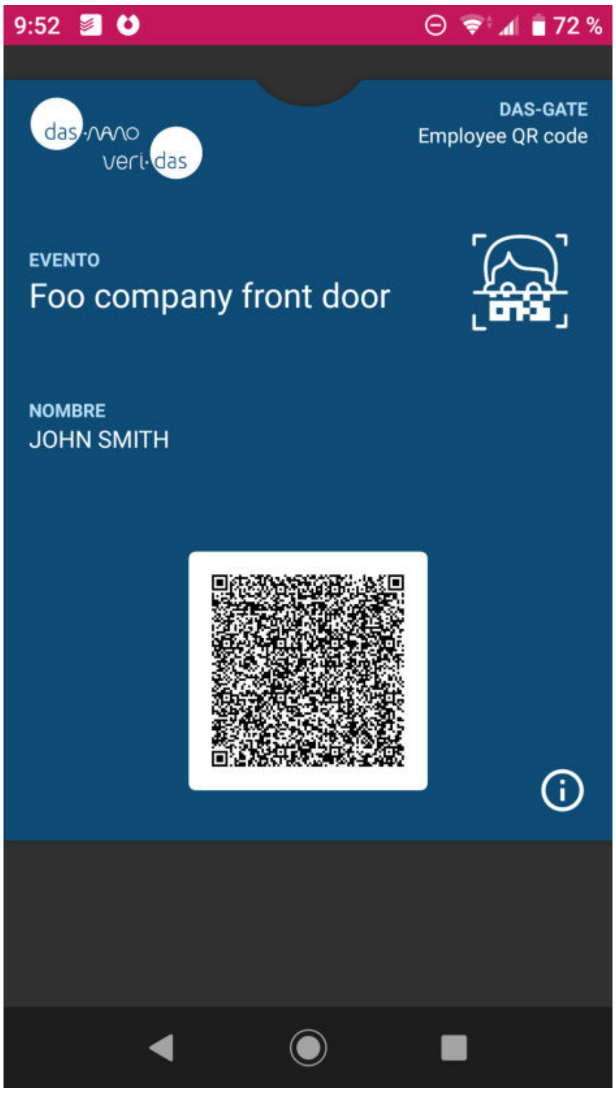
das-FaceQR is a service which allows the authentication of a person via face biometry. This service is provided as a cloud-based or SaaS solution that can be consumed via APIs. Unlike das-Face -another Veridas' face biometry service which, among others, allows to compare face images-, the process of biometric verification is performed as a comparison between a face image and an abstract representation of the face of the person stored in a biometric credential.

The face biometric credential is a mathematical descriptor obtained from the characteristics of the face in a face photo. This mathematical conversion from the face into a biometric vector is irreversible. Therefore, it is not possible to recover a person's face from the calculated biometric vector.

All the information needed to verify the identity of the bearer of the credential is contained in the credential itself. Veridas does not keep any information either when generating a credential or verifying it against the bearer's face. These credentials are secure, as it is not possible for adversaries to modify their content without such modification being later undetectable.

A credential could be bundled in a QR code rendered in an image, or in a Passbook file. The latter is readable by many of the wallet applications available in the market for almost any mobile operating system. A screenshot of a Passbook file generated with dasFaceQR is shown below.

Section 5.3 outlines all the parameters that can be used to tweak the captions and the contents that will be shown in the wallet.



2. dasFace QR 2021Q2 What's new

2.2. Changed

- Aztec Codes can now be generated without base85 encoding and without custom headers which made that can be readed by any conventional Aztec code reader

3. dasFace QR main features

This section presents the technical features and differentials of das-FaceQR. These differentials are mainly based on the concepts of security and privacy.

3.1. Security

The security of the das-FaceQR service is based on 4 principles.

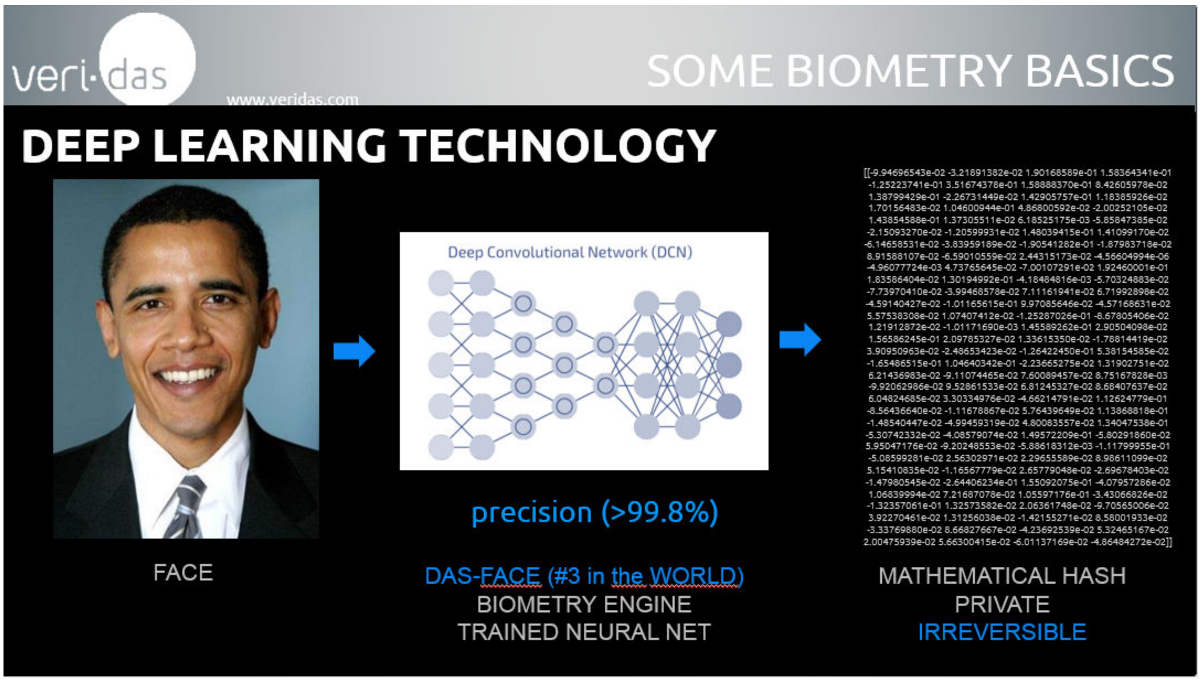
1. High performance biometric engine.
2. Encryption of biometric information.
3. Valueless authentication.
4. Use of double factor authentication.

3.1.1. *High performance biometric engine*

dasFaceQR leverages a top-notch high-performance biometric engine to generate the biometric information associated with a person. This engine has been developed by Veridas from scratch using the latest Artificial Intelligence technology, and was ranked as the third best biometric engine in the world by NIST on April 4, 2019.

3.1.2. *Encryption of the biometric information and digital signature*

The biometric QR generation process allows obtaining an encrypted mathematical representation of a person's face as a string of bytes. This means that anyone managing to read one these credentials will not be able to obtain any personally-identifiable information out of it.



```
[[-9.94696543e-02 -3.21891382e-02 1.90168589e-01 1.58364341e-01
 -1.25223741e-01 3.51674378e-01 1.58888370e-01 8.42605978e-02
 1.38799429e-01 -2.26731449e-02 1.42905757e-01 1.18385926e-02
 1.70156483e-02 1.04600944e-01 4.86800592e-02 -2.00252105e-02
 1.43854588e-01 1.37305511e-02 6.18525175e-03 -5.85847385e-02
 -2.15093270e-02 -1.20599931e-02 1.48039415e-01 1.41099170e-02
 -6.14658531e-02 -3.83959189e-02 -1.90541282e-01 -1.87983718e-02
 8.91588107e-02 -6.59010559e-02 2.44315173e-02 -4.56604994e-06
 -4.96077724e-03 4.73765645e-02 -7.00107291e-02 1.92460001e-01
 1.83586404e-02 1.30194992e-01 -4.18484816e-03 -5.70324883e-02
 -7.73970410e-02 -3.99468578e-02 7.11161941e-02 6.71992898e-02
 -4.59140427e-02 -1.01165615e-01 9.97085646e-02 -4.57168631e-02
 5.57538308e-02 1.07407412e-02 -1.25287026e-01 -8.67805406e-02
 1.21912872e-02 -1.01171690e-03 1.45589262e-01 2.90504098e-02
 1.56586245e-01 2.09785327e-02 1.33615350e-02 -1.78814419e-02
 3.90950963e-02 -2.48653423e-02 -1.26422450e-01 5.38154585e-02
 -1.65486515e-01 1.04640342e-01 -2.23665275e-02 1.31902751e-02
 6.21436983e-02 -9.11074465e-02 7.60089457e-02 8.75167828e-03
 -9.92062986e-02 9.52861533e-02 6.81245327e-02 8.68407637e-02
 6.04824685e-02 3.30334976e-02 -4.66214791e-02 1.12624779e-01
 -8.56436640e-02 -1.11678867e-02 5.76439649e-02 1.13868818e-01
 -1.48540447e-02 -4.99459319e-02 4.80083557e-02 1.34047538e-01
 -5.30742332e-02 -4.08579074e-02 1.49572209e-01 -5.80291860e-02
 5.95047176e-02 -9.20248553e-02 -5.88618312e-03 -1.11799955e-01
 -5.08599281e-02 2.56302971e-02 2.29655589e-02 8.98611099e-02
 5.15410835e-02 -1.16567779e-02 2.65779048e-02 -2.69678403e-02
 -1.47980545e-02 -2.64406234e-01 1.55092075e-01 -4.07957286e-02
 1.06839994e-02 7.21687078e-02 1.05597176e-01 -3.43066826e-02
 -1.32357061e-01 1.32573582e-02 2.06361748e-02 -9.70565006e-02
 3.92270461e-02 1.31256038e-02 -1.42155271e-02 8.58001933e-02
 -3.33769880e-02 8.66827667e-02 -4.23692539e-02 5.32465167e-02
 2.00475939e-02 5.66300415e-02 -6.01137169e-02 -4.86484272e-02]]
```

It is not possible to retrieve a person's face image from the biometric vector. In other words, the mathematical operation that transforms a face into a biometric vector is irreversible (assimilable to a “hash”).

The encryption of the biometric vector resulting in the biometric credential provides an additional layer of security. To do this, a 32-byte key is used. Currently, the encryption process uses a unique key, known exclusively to Veridas.

Why is the biometric vector encrypted if the vector itself is an irreversible byte string?

Additionally, multi-tenant encryption is necessary to prevent the customers (with access to the on-premises instance of das-FaceQR) from using the biometric QRs of other customers. Currently, Client A can take a biometric QR generated by Client B and perform a biometric validation process if they have an on-premises instance of das-FaceQR. In any case, Client A and Client B can not access the user's biometric information.

3.1.3. *Value-less authentication*

das-FaceQR introduces the concept of “valueless authentication” or value-less authentication. This concept refers to the null importance of the loss of the biometric QR.

- No one -a person or an automatic system-, even Veridas, can recover the user's facial image.
- Only the Veridas software can retrieve the biometric vector
- From the customer's point of view, the loss of the biometric QR does not expose any personal data of its users (except in the case that they are added as part of the contextual data

3.1.4. Double factor authentication

Currently there are different authentication mechanisms in applications. In general, they use at least one of the following elements.

- Something the user knows: It is the usual case of authentication by email and password. In this scenario, the application stores these two data, where the password is known -ideally-exclusively by the user. In this situation, the application has access to user information, for example the email, which does not have to be necessary for the consumption of the service offered by the application.
- Something that you are: It is the usual case of biometric authentication (facial, fingerprint, etc.). In this scenario, the application stores the registration biometric information, in the

form of an image, biometric vector, or other. As in the previous case, the application has access to user information, for example the image of the face, which does not have to be necessary for the consumption of the service offered by the application.

- Something you have: Usually used as a complement to the previous two, with the aim of reinforcing security.

das-FaceQR is based on the use of two factors combined.

- Something that you are, through the use of facial biometrics.
- Something you have, by storing the biometric credential by the user in physical or digital format.

The validation through the das-FaceQR service implies that the user is who he claims to be while presenting something he has.

Likewise, if das-FaceQR is used in the context of digital payment services, the legal requirements established by the European PSD2 regulations could be met for considering that a "stronger authentication" has taken place. The element of inherence (something you are) is fulfilled by the biometric verification. This can be completed and converted in a reinforced authentication by taking the biometric credential held by the user as a token for the purposes of compliance with the element of possession (something you have).

3.2. Privacy

The main feature and differential element of das-FaceQR is that it offers the possibility of implementing an authentication system in which the application that the user uses to authenticate, does not have to store any personal (or biometric) data of the user.

Unlike the previously explained authentication mechanisms, the registration information can be guarded (stored) by the user, not by the application. As the biometric credential has no value itself, it can be stored by the user without limitation and risk free, allowing a qualitative leap in terms of the privacy of the user data.

3.3. Output formats

The code containing the biometric credential can be exported in several formats. For instance, it could be an image containing the QR or Aztec code, or a Passbook file readable with any of the widely available wallet applications.

3.4. Selfie image formats

Admitted formats for the selfie images are JPEG, PNG and TIFF.

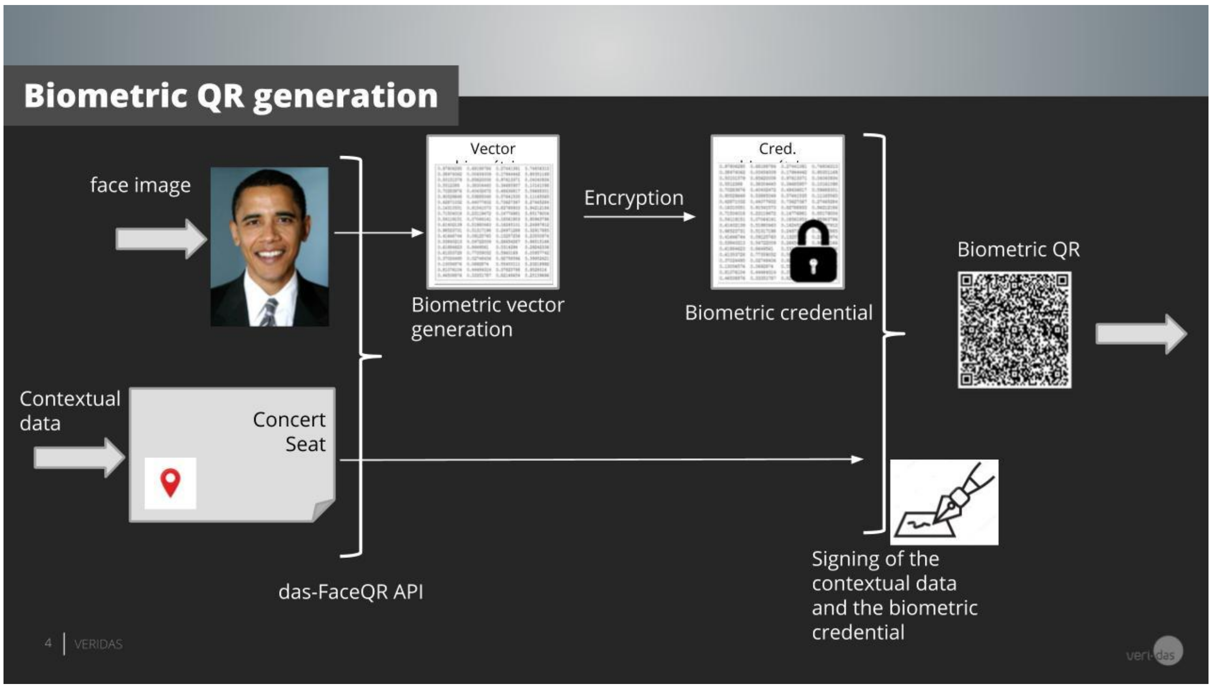
4. Operation details

This section describes the logic and details which happen behind the different das-FaceQR features which are offered to be consumed as an API.

4.1. Creation of the Biometric Credential

From a facial image, das-FaceQR returns an abstract representation of the face in a QR format. This is called biometric credential. The process of biometric credential generation is the following.

1. Some data is sent to the das-FaceQR credential generation endpoint. This data will typically be a picture with a face, and some arbitrary data referred to as “contextual data”, though both are optional.
2. Behind the scenes, das-FaceQR calls das-Face (the Veridas’ face biometric engine offered as an API) which returns a biometric vector. Currently, the size of the vector is 1872 bits, to save as much space as possible.
3. Additionally, das-Face adds useful information of the biometric generation process. This information includes a timestamp and a hash with the version of the biometric engine which generated the vector.
4. das-Face carries an encryption process with this information, producing a biometric credential. This credential has a size of 1872 bits. Optionally, das-FaceQR admits that the customer includes additional information related to the use case. This information is known as contextual data.
5. das-FaceQR signs the biometric credential and the contextual data.
6. das-FaceQR incorporates the previous information in a compact representation of the information, based on the QR standard. This is known as a biometric QR.
7. The credential is returned as an image with a QR code, or a Passbook file, as chosen by the caller.

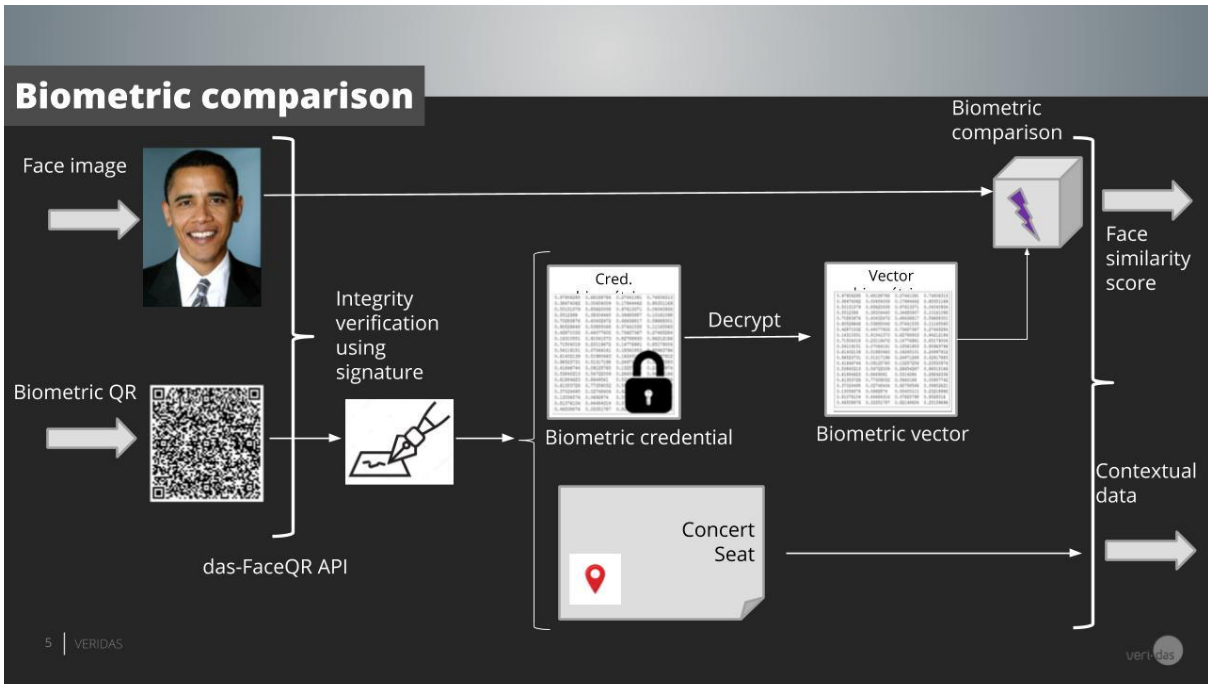


4.2. Biometric Authentication

By using a face image and a biometric credential, the biometric authentication process is carried on, obtaining as a response a face similarity value and the contextual data that were introduced in the biometric credential during its generation.

1. The returned QR code is read out-of-band later by standard QR scanning hardware, which is widely available elsewhere. The raw data read from the QR code is submitted to the credential verification endpoint, together with a selfie image.
2. First of all, das-FaceQR validates the integrity of the data set stored in the biometric QR, by using for that the signature introduced in the generation process. If the biometric QR integrity can not be guaranteed, das-FaceQR returns an API error.
3. If the biometric QR integrity can be granted, das-FaceQR obtains the biometric credential and decrypts it, obtaining the biometric vector.
4. **das-FaceQR** calls to the Veridas' biometric engine (das-Face) to biometrically compare the face image that was introduced and the decrypted biometric vector. A similarity between both face representations is obtained. The similarity value is a float number with a value ranging from 0 and 1. As greater the number, greater the probability of the two representations to belong to the same person.
5. Finally, das-FaceQR obtains the contextual data stored in the QR and returns them through the API alongside the face similarity value previously calculated. The similarity score tells how likely it is that the submitted selfie is of the same person as the one that was used to

generate the QR. If no biometric data was found in the credential (because no selfie was used to generate the QR) then the similarity score will be zero.



The face images used in the creation and in the biometric authentication must have at least 100 pixels of distance between the eyes of the face of the image person. It is recommended to use the Veridas capture SDKs available for iOS, Android and HTML platforms.

An specific use case example is included in the *Annex B* of the current document.

5. API Considerations

This service exposes an API with the following features.

Requests must meet the following requirements:

- Content type must be either multipart/form-data, or application/json.
- The API can only be accessed via TLS. Always validate server certificates and never trust a VeriSaaS endpoint that offers an invalid certificate.

The documentation for all the endpoints that require a POST method has been written assuming a multipart/form-data MIME type. However JSON is equally accepted for all such endpoints and the behavior should be the same.

The following differences should be taken into account when consuming an API with a JSON body:

- Content type must be set to application/json. Otherwise the API will assume a multipart body.
- All the parameters that have been marked as body parameters in the endpoints' descriptions (location: body) can be sent in the JSON body. For example, in the /credential/qr-code endpoint, the parameters selfieImage and contextualData are marked as such. Hence, these parameters can also appear in the JSON object.
- Query string parameters will continue to be so regardless of the body format.
- Mandatory parameters (marked as 'required') will continue to be so regardless of the body format.
- **All JSON values MUST be encoded in base64**, whereas multipart bodies need not, except where explicitly stated. This is a relevant detail when sending binary data.

Some endpoints require a biometric model to be specified. These endpoints have been conveniently marked with the placeholder {model}, as this parameter is passed in the path.

Example:

```
/dasfaceqr/v2/3a9e9d5ffd5de4c212c2aff26eeca523fb69754e604894520b32e4ed/credential/qr-code
```

Models can be specified either by their hash (as in the example above) or by a tag. An example of a model tag is given below.

```
/dasfaceqr/v2/20190813/credential/qr-code
```

Available models can be obtained with the GET /models endpoint.

6. API Definition

Public Base URL:

https://<base_url>/dasfaceqr/v2

Resources:

Method	**Public URL**	**Description**
GET	/alive	Check if the service is up. Should reply with a 204 No Content status code.
GET	/models	Get the available models and their identifiers.
POST	{model}/credential/qr-code	Generate a QR or Aztec code with the

	{model}/credential/aztec-code
	{model}/credential/qr-code/algorithm/{algo name}
	{model}/credential/aztec-code/algorithm/{algo name}
		supplied selfie image and/or contextual data.

POST	{model}/credential/qr-code/passbook/event {model}/credential/aztec-code/passbook/event	Generate a Passbook file suitable for opening in wallet applications. Will contain a QR or Aztec code with the supplied selfie image and optional contextual data.
POST	/verification/credential/qr-code /verification/credential/aztec-code	Compare an image with the biometric vector contained in a QR or Aztec code.
GET	/certificate/{algorithm}	Obtain the certificate to verify the generated signatures. May be followed by an algorithm identifier.

POST	/qr-code/unsigned
	/aztec-code/unsigned
	{model}/qr-code/unsigned
	{model}/aztec-code/unsigned
	/qr-code/algorithm/{algo name}
	/aztec-code/algorithm/{algo name}
	{model}/qr-code/algorithm/{algo name}
	{model}/aztec-code/algorithm/{algo name}
		Generate a QR or Aztec code with a standard format. This means that the provided data (selfie and/or contextual data) will be introduced in the QR as “Raw data” (without any additional bytes).

POST	/verification/qr-code /verification/aztec-code /verification/qr-code/unsigned /verification/aztec-code/unsigned	Extract the content of the QR or Aztec code. If a selfie is provided, it will compare it with the biometric vector extracted from the QR or Aztec code. Otherwise, the content of the QR or Aztec code will be returned as pure contextual data.

6.1. Check if the service is alive

GET /alive

Response: HTTP status 204 No Content.

Check if the service is up.

6.2. Get the available biometric models

GET /models

Response: HTTP status 200 OK

Get a list of the supported biometric models. The list is a JSON object where each item identifies a model. Each item, in turn, will have two keys: a hash and a tag. Either of these can be used in the {model} placeholder for the API endpoints that require it. An example is provided below.

```
[
        {
        "tag": "20200514",
        "hash": "904fa9ef6e71ef541f20a95d3dc97821b7af43b8cd2c1bb3eb09df15"
        },
        {
        "tag": "20190813",
        "hash": "3a9e9d5ffd5de4c212c2aff26eeca523fb69754e604894520b32e4ed"
        },
        {
        "tag": "20180827",
        "hash": "b0bf475e5344e816f12b83c13c075a3256ef95e60ac1cdc273aef59f"
        }
]
```

6.3. Generate a QR code

POST /{model}/credential/qr-code
POST /{model}/credential/qr-code/algorithm/{algo name}
POST /{model}/credential/qr-code/passbook/event
POST /{model}/credential/qr-code/passbook/event/algorithm/{algo name}
POST /{model}/credential/qr-code/unsigned
POST /{model}/credential/aztec-code
POST /{model}/credential/aztec-code/algorithm/{algo name}
POST /{model}/credential/aztec-code/passbook/event
POST /{model}/credential/aztec-code/passbook/event/algorithm/{algo name}
POST /{model}/credential/aztec-code/unsigned

Response: HTTP status 200 OK, and an image or Apple Passbook file (application/vnd.apple.pkpass).

Generate a QR or Aztec code with the supplied selfie image and contextual data. Both are optional, but at least one of them MUST be provided. Either selfie image only, or contextual data only, or both.

The last variant (with the 'unsigned' suffix) generates unsigned credentials. Do not use this variant unless you are absolutely sure of what you are doing, as it is very easy for a malicious user to forge unsigned credentials.

This endpoint requires a model only if a selfie image is provided, since a biometric engine is needed to convert the selfie image to a biometric vector. The model identifies the specific biometric engine to use. If no selfie image is provided (contextual data only) then the {model} placeholder MUST be omitted. If a model is specified but no selfie image provided, a 400 error will be returned.

Returns a QR code image or an Apple Passbook file. At the time being, only event type Passbooks are supported.

It is not currently possible to request an Apple Passbook file with the Accept header only. Passbook files can be generated by appending /passbook/event to the path.

The data in the QR is composed of the biometric vector, the contextual data and a digital signature. But as has been noted above, the signature and the biometric vector can be omitted if desired. In that case, the QR will just contain the contextual data, and a small header consisting of two control bytes. The public key for verifying it can be obtained with the /certificate endpoint.

It is however possible to prepend an arbitrary string of data to the "normal" content of the QR code. This data needs to be passed in the prefixData parameter. This data will be put verbatim in the first bytes of the QR data, and will NOT form part of the signed data. This means anyone that gets hold of the QR code can freely modify this part without dasFace QR being able to detect later on at the verification phase.

The default signature algorithm that will be used to sign the QR code is ISO/IEC 9796-2, also known as "RSA with message recovery". This is the algorithm that will be applied if no specific algorithm is specified. An algorithm can be explicitly selected by appending /algorithm/{algo name} to the qr-code or aztec-code endpoint. The following table lists the supported algorithms. The {algo name} placeholder would need to be replaced by one of the names in the first column.

Algorithm identifier	**Description**
`iso9796-2-2010` or just `iso9796-2`	ISO/IEC 9796-2 algorithm (2010 revision), also known as RSA with message recovery. This generates the smallest signatures and is the algorithm used by default. Though the other algorithms are a bit more secure.
`ed25519`	ed25519 algorithm, which generates 64-byte signatures. These are the smallest ones available with an acceptable security level.
`ed448`	ed448 algorithm. This is a high-security algorithm that generates slightly bigger signatures.

It is possible to specify the QR code version and error correction level. Please take into account that different versions have different size constraints. The limits explained in the above paragraph apply to version 28 and error correction M, which are the default values. Accepted versions range from 12 to 32 (both included). An endpoint is provided that allows querying these size limits (see section 1.4). Annex A lists all the supported QR versions and the maximum number of bits of contextual data that can be put into them.

Sending binary contextual data

Binary contextual data is supported, but it MUST be sent as if it was a file (with a filename attribute and a multipart/form-data MIME type). The value of the filename attribute is irrelevant, but the parameter itself must exist. Otherwise the data will be taken as printable and that will cause 400 errors because of the wrong encoding of binary data.

Content of Apple Passbook files

In addition to the QR code with the biometric vector and contextual information, Passbooks also contain some additional information for the user. This information is displayed on the device's screen when the Passbook is read, and is organized in a number of fields. This information is not stored in the QR code.

Standard parameters

The following parameters are accepted, for both endpoints.

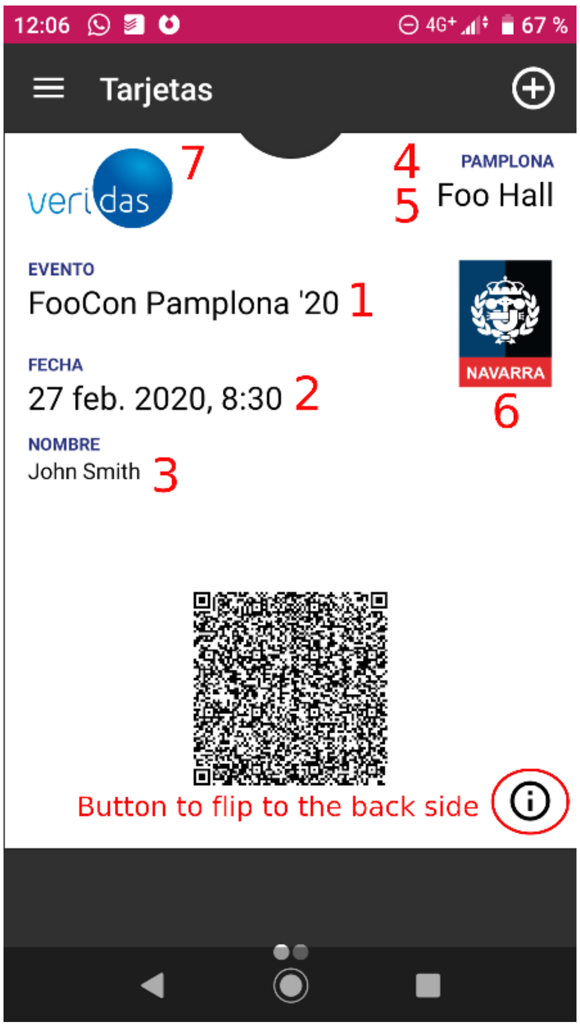
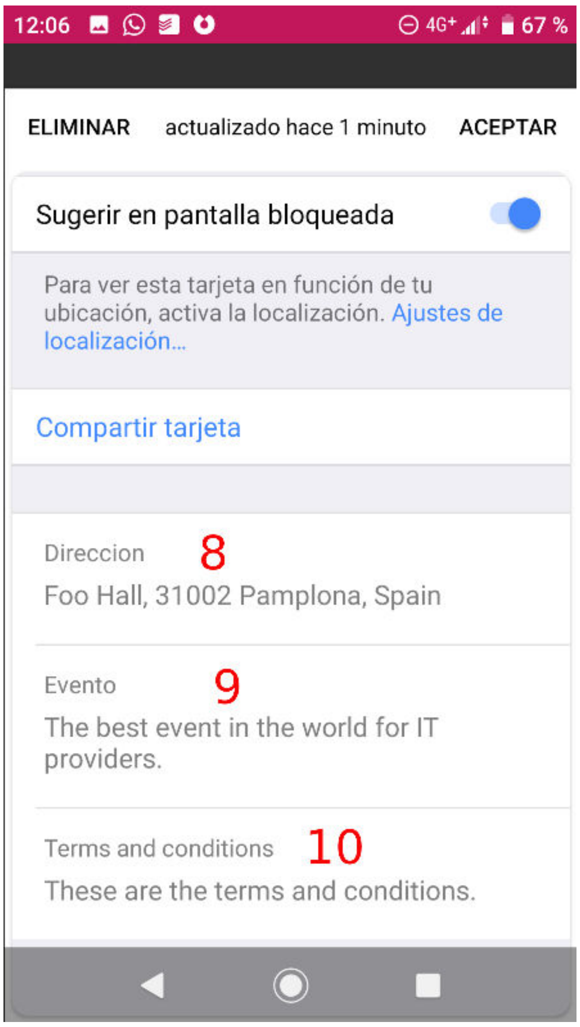
Parameter name	Location	Description
`selfieImage`	Body	A picture with a selfie. Refer to section 3.4 to see a list of admitted formats.
`contextualData`	Body	Some optional contextual information. Binary data is allowed but there are some constraints (see section "Sending binary contextual data").
`model`	Path	Identifier of the model that will be used to extract the face features and generate the biometric vector. If a selfieImage was provided, then a model MUST also be provided.
`prefixData`	Body	An arbitrary string of bytes that will be put just before the dasFaceQR-generated content in the QR code. This data will not be signed.

Parameters for wallet Passbook files

The following parameters can be passed to the Passbook file generation endpoint (`/passbook/event`). Each of these parameters controls one aspect of the Passbook's visual appearance. Two screenshots are also shown, taken on an Android device, displaying a sample Passbook file's front and back. Each visual item (event name, date, etc.) is labeled with a number, and referenced in the parameter that controls it.

Parameter name	Required	Location	Description
`latitude`	yes	Body	Latitude of the coordinates where the event will be taking place.
`longitude`	yes	Body	Longitude of the coordinates.
`title`	no	Body	Most Passbook readers will also show this as a simplified description of the Passbook on a list of items. Item labeled 4 shows the title.
`subtitle`	no	Body	Most Passbook readers will also show this (together with the `title`) as a simplified description of the Passbook on a list of items. Item labeled 5 shows the subtitle.
`eventName`	no	Body	The name of the event or place. This should be a short phrase, as it will be prominently displayed at the top. Item labeled 1 shows the event name.
`personName`	no	Body	Name of the individual. Ideally this should be the person whose biometric credential has been put in the enclosed QR code (parameter `selfieImage`).
`address`	no	Body	Address where the event will be taking place. Will be shown as a secondary field. Item labeled 8 shows the address.
`description`	no	Body	A text describing the event. Will be shown as a secondary field. Item labeled 9 shows the description.
`termsAndConditions`	no	Body	A terms-and-conditions text. Will be shown as a back field. Item labeled 10 shows the terms and conditions.
`date`	no	Body	Date (and optionally, time), at which the event will be taking place, in

			standard ISO format (i.e. YYYY-mm-dd HH:MM:ss). The time part can be skipped, as well as the seconds. The date will and displayed in a human-readable format, and localized (i.e. in the user’s language). Item labeled 2 shows the date.
`logo and logox2`	no	Body	An image that will be displayed on the front of the Passbook in the top left corner. The field logox2 should have the same image that has been put in the logo (though this is not enforced), but in a bigger size. Item labeled 7 shows the logo.
`icon and iconx2`	no	Body	An image that will be displayed in notifications related to this Passbook that the device (i.e. a phone) might pop up to the user. The relationship of iconx2 with regards to icon is the same as with logo and logox2 - it should contain the same image in a bigger size.
`thumbnail`	no	Body	An additional image displayed on the front of the Passbook. For example, this could be a picture of the cardholder. Item labeled 6 shows the thumbnail.
`labelColor`	no	Body	Color of the text for the labels’ captions, in HEX format (e.g. #ffaabb). Default is dark blue (#2d3581).
`foregroundColor`	no	Body	Color of the rest of the text, in HEX format. The default is black.
`backgroundColor`	no	Body	Background color, in HEX format. The default is white.

Front	Back
	

Front:

12:06 · 67 %

Tarjetas

7 · 4 PAMPLONA · 5 Foo Hall

EVENTO
FooCon Pamplona '20 1

FECHA
27 feb. 2020, 8:30 2

NOMBRE
John Smith 3

NAVARRA 6

Button to flip to the back side

Back:

12:06 · 67 %

ELIMINAR · actualizado hace 1 minuto · ACEPTAR

Sugerir en pantalla bloqueada

Para ver esta tarjeta en función de tu ubicación, activa la localización. Ajustes de localización...

Compartir tarjeta

Direccion 8
Foo Hall, 31002 Pamplona, Spain

Evento 9
The best event in the world for IT providers.

Terms and conditions 10
These are the terms and conditions.

Optional parameters for image QR

The following optional parameters can be passed to the QR generation endpoint (image format).

Parameter name	Location	Description
`qrVersion`	Query	QR version to generate. Accepted versions are from 12 to 32 (both included). Default version will be 28.
`qrErrorCorrection`	Query	Error correction level to use, as specified by the QR standard. Accepted values are L, M, Q and H. Default error correction is M.
`boxSize`	Query	Controls how many pixels each box of

		the QR code is. The default value (if omitted) is 12. Allowed values are 4, 12 and 24.
`binary`	Query	QR format. The default value is ‘false’, which means that the endpoint will return the QR as an image. If ‘true’ is given, the endpoint will return the content of the QR as raw bytes.

Responses

Status code	**Description**
200 OK	Success. An image or Passbook file should be expected in the response body.
413 Request Entity Too Large	Submitted selfie image and/or contextual data are too large.

400 Bad Request	A mandatory parameter was omitted, or some invalid value was supplied. The body should contain a JSON with extended information about the error.
	
	Example responses:
	
	`{`
	`"code": "FormatError",`
	`"message": "SubjectPublicKeyInfo does not match public key in X.509 certificate"`
	`}`

6.4. Verify a QR code, with similarity comparison

POST /verification/credential/qr-code
POST /verification/credential/qr-code/unsigned
POST /verification/credential/aztec-code
POST /verification/credential/aztec-code/unsigned

Response: 200 OK, and a JSON (application/json) in the body.

Compare an image with the biometric vector contained in a QR or Aztec code.

There are separate endpoints to verify QR codes and Aztec codes. This is so because, even though they look quite similar, their structure, character set and encoding are very different. This will be consistent with virtually all client setups, as the reader will always know whether it's reading a QR code or an Aztec code. However, once the payload has been read, DasFaceQR cannot reliably guess whether it comes from QR or Aztec.

This endpoint takes the data read from a QR code and a selfie image, and compares the selfie image with the biometric credential contained in the QR. It returns a confidence score (how likely it is both selfies are of the same person) and the contextual data contained in the QR code, if any. The contextual data is always returned encoded in base64.

If no biometric credential is present in the QR/Aztec code, then the provided selfie image will be ignored (you may even not provide any), and the confidence score will always be zero.

In the case a biometric credential is found in the credential but no selfie image was provided for comparison, a 400 Bad Request error code will be returned.

If some data was prepended to the credential when it was generated (using the prefixData parameter, see endpoint /credential/qr-code) then this data needs to be removed by the caller BEFORE calling this endpoint. Not doing so will cause the signature verification to fail. dasFaceQR will not attempt to "detect" the presence of caller-introduced prefix data in any way.

Unsigned credentials need to be verified with the second endpoint (the one with the 'unsigned' suffix). Other than the fact that it can verify unsigned credentials, its behavior is identical to the first endpoint. Trying to verify a signed credential with this endpoint will result in an error, as signed and unsigned credentials have differing formats.

A note on reading QR codes

The QR codes generated by dasFace QR contain binary information. The main reason for this is to save space.

Some libraries try to convert binary data to printable strings automatically, unless overridden by the caller, sometimes in a non intuitive way. We have observed this behavior in the ZXing library for JavaScript, for instance. Please bear this in mind when reading one of these QR codes.

For example, if you use JavaScript's ZXing library, you need to access the resultMetadata field to obtain the raw binary data contained in the QR code.

```
const bytes = qrdata['resultMetadata'];
```

Parameters
The following parameters are accepted.

Parameter name	Required	Location	Description

`selfieImage`	no	Body	A picture with a selfie. The maximum admitted size for the image is 2 MB. Refer to section 3.4 to see a list of admitted formats.
`qrData`	yes	Body	Data read from a QR code, base64-encoded. QR codes contain binary data that must be encoded in base64. Bear in mind that some libraries might make it hard to treat binary data, as explained in the section above "A note on reading QR codes".

Responses

Status code	**Description**

200 OK	Success. A JSON object should be expected in the response body. This JSON object contains the following fields:
	
	● confidenceNumber: A score between 0 and 1. A higher value means there is a higher probability that the person in the selfie image is the same as the person of the read biometric credential (QR code).
	● contextualData: Contextual data taken verbatim from the QR data, which was supplied in the previous endpoint /credential/qr-code. This information is always returned base64-encoded.
	
	Example response:
	{
	"confidenceNumber": 0.018946728477870274,
	"contextualData":
	"IGZhc2RkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGR
	kZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZG
	RkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZ
	GRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRk
	ZGRkZGRkZGRkZA=="
	}

415 Unsupported Media Type	The supplied credential is invalid.
400 Bad Request	A mandatory parameter was omitted, or some invalid value was supplied. The body should contain a JSON with extended information about the error. This error is also returned when the signature of the supplied QR data could not be verified.

	Example responses:
	
	`{`
	`"code": "InvalidInputError",`
	`"message": "Parameter 'selfieImage' was not supplied"`
	`}`
	
	`{`
	`"code": "FormatError",`
	`"message": "SubjectPublicKeyInfo does not match public key in X.509 certificate"`
	`}`

6.5. Obtain the certificate to verify a QR code's signature

GET /certificate/{algorithm}

Response: HTTP status 200 OK.

Obtain the certificate to verify the generated signatures, for the given signature algorithm.

This endpoint returns the certificate (in PEM format) with the public key that can be used to verify the signatures of the QR codes generated by dasFace QR.

The algorithm placeholder needs to be replaced with the name of a supported algorithm. These are listed in the table at section 4.3.

If omitted, the certificate for the default algorithm (ISO/IEC 9796-2) will be returned.

Parameters

This endpoint takes no parameters.

Responses

Status code	**Description**
200 OK	Success. An X.509 certificate encoded in PEM format (application/x-x509-ca-cert) should be expected in the body.

6.6. Generate a standard QR code

POST {model}/qr-code/unsigned
POST {model}/qr-code/algorithm/{algo name}

POST {model}/qr-code/passbook/event/algorithm/{algo name}
POST {model}/aztec-code/unsigned
POST {model}/aztec-code/algorithm/{algo name}
POST {model}/aztec-code/passbook/event/algorithm/{algo name}

Response: HTTP status 200 OK, and an image or Apple Passbook file (application/vnd.apple.pkpass).

This endpoint works similarly to the previously explained endpoint (6.3) with a few differences.

The generated QRs have a standard encoding format (see link for more information). This means that the content of the QR is formed only by the desired data (contextual data and/or biometric vector).

If the url ends with /unsigned the generated QR will only contain the provided data. Otherwise, if the url ends with /algorithm/<algorithm_name> the content of the QR will be signed (with the specified algorithm). The supported encryption algorithms are: ed25519 and ed448.

Parameters
The parameters needed for this endpoint are the same as in 6.3 but excluding "prefixData", which doesn't apply in this context.

Responses
The response format is the same as in 6.3.

6.7. Verify a standard QR code

POST /verification/qr-code
POST /verification/qr-code/unsigned
POST /verification/aztec-code
POST /verification/aztec-code/unsigned

Response: 200 OK, and a JSON (application/json) in the body.

This endpoint tries to verify a QR with a standard format. If a selfie is provided and a biometric vector is found in the QRs content, it will compare them and return a confidence score with the contextual data extracted from the QR (if any). If no selfie is supplied, it will assume that the QR contains only contextual data and it will return it as an encoded base64 string.

By default, this endpoint will assume that the QR is signed and it will try to verify its content. If a non signed QR wants to be verified, "/unsigned" needs to be appended to the path.

Parameters

The parameters needed for this endpoint are the same as in 6.4 but excluding “prefixData”, which doesn’t apply in this context.

Responses
The response format is the same as in 6.4.

7. API Errors

The API defines the following error messages.

Code	**HTTP Status**	**Description**
FaceError	400	das-FaceQR was unable to locate a person's face in the supplied picture. The attached message should provide more details of what went wrong.
SignatureError	400	Signature verification failed.
FormatError	400	The supplied binary blob does not follow the formatting rules expected by das-FaceQR. Most of the time this would indicate that the supplied data is malformed in some way. We have observed some QR reading libraries try to decode the binary data, which may also cause this error. See section “A note on reading QR codes”.
InvalidInputError	400	A required parameter is missing, or has an invalid value.
ContextualDataTooLargeError	413	Submitted contextual data is too large.
InvalidEncoding	400	The supplied `qrData` has an unexpected encoding format. This error may occur after sending a QR code to the Aztec code endpoint and vice-versa.
RequestBodyNotFound	400	The request body is empty.
UnsupportedCodeType	400	The supplied code type (QR, aztec, etc.) is not supported.
InconsistentQRContent	400	The provided qr-data has inconsistent information. For instance: the data says that the qr is 100 bytes long, but it’s not.
UnsupportedModel	400	The provided Model is not supported.

Annex A: Supported QR versions and maximum contextual data lengths

The following two tables show the maximum length of contextual data that can be stored for each QR code version and redundancy level. All these values have been calculated supposing a selfie has also been sent, and hence a biometric vector is also stored in the QR code (which takes up additional space). Those combinations that cannot be used (such as 12 M) because they don't have enough capacity for just the biometric vector, have been omitted from the tables.

For ed25519 algorithm

QR version and error correction level	Max. length of contextual data (bytes)
12 L	40
13 L	98
13 M	4
14 L	131
14 M	35
15 L	193
15 M	85
16 L	259
16 M	123
17 L	317
17 M	177
18 L	391
18 M	233
19 L	465
19 M	297
20 L	531
21 L	602

21 M	384
22 L	676
22 M	452
23 L	764
23 M	530
24 L	844
24 M	584
25 L	946
25 M	670
25 Q	388
26 L	1040
26 M	732
26 Q	424
27 L	1138
27 M	798
27 Q	478
28 L	1201
28 M	863
28 Q	541
29 L	1301
29 M	937
29 Q	581
30 L	1405
30 M	1043
30 Q	655
30 H	415

31 L	1513
31 M	1125
31 Q	703
31 H	463
32 L	1625
32 M	1211
32 Q	785
32 H	515

For ISO/IEC 9796-2 algorithm

QR version and error correction level	Max. length of contextual data (bytes)
12 L	47
13 L	105
13 M	11
14 L	138
14 M	42
15 L	200
15 M	92
16 L	266
16 M	130
17 L	324
17 M	184
18 L	398
18 M	240
19 L	472
20 L	538

21 L	609
21 M	391
22 L	683
22 M	459
23 L	771
23 M	537
24 L	851
24 M	591
25 L	953
25 M	677
25 Q	395
26 L	1047
26 M	739
26 Q	431
27 L	1145
27 M	805
27 Q	485
28 L	1208
28 M	870
28 Q	548
29 L	1308
29 M	944
29 Q	588
30 L	1412
30 M	1050
30 Q	662

30 H	422
31 L	1520
31 M	1132
31 Q	710
31 H	470
32 L	1632
32 M	1218
32 Q	792
32 H	522

Annex B. Use case example: Authentication for accessing a concert

Here it is described a complete use case of the das-FaceQR technology in combination with other Veridas digital authentication products. The example presents a concert ticket sales process and an authentication process in the event entrance.

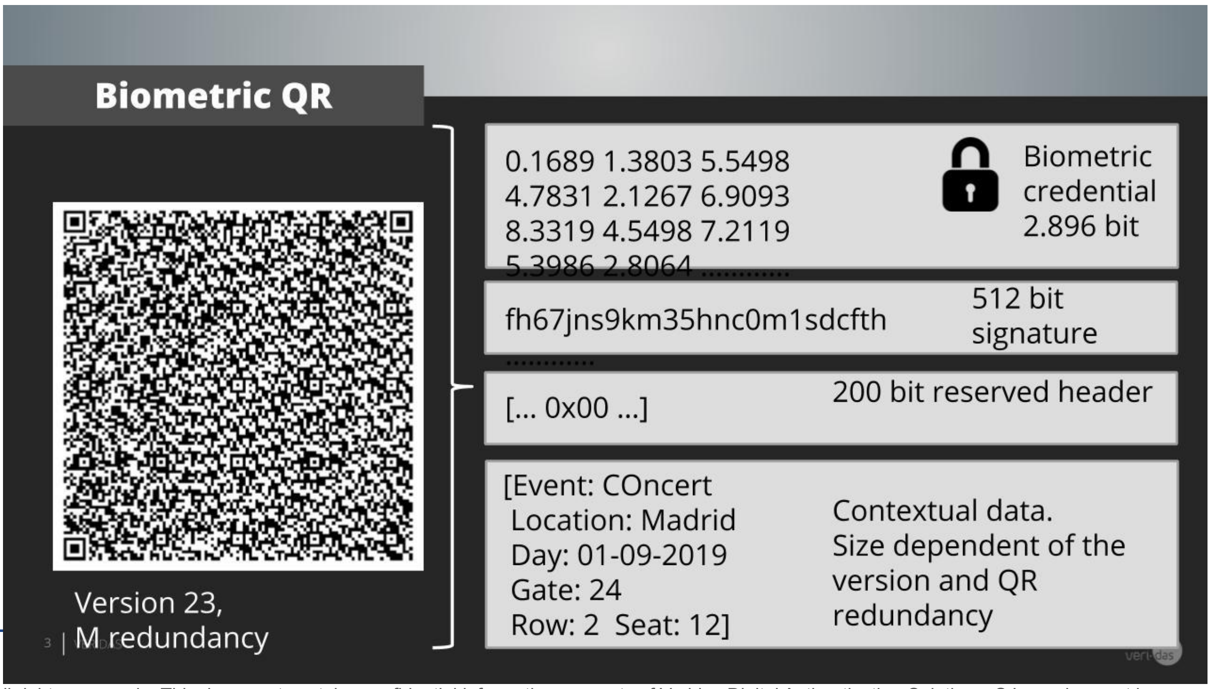
Tickets sale

Someone decides to go to a concert. To do that, a ticket has to be bought. The chosen channel to buy it is the tickets sale webpage so it will be done remotely. During the buying process, certain personal information is requested to the client, like its name, surnames, etc. as well as its banking information. Veridas does not take part in any of these processes.

Once these forms are completed and submitted, the tickets sale webpage uses the Veridas selfie-alive SDK and requests the user to do a selfie photo. This SDK requests the user to do an action to continue the process, using the action to verify that the person is alive. Currently, the requested action is to smile.

After the capturing process is completed, the service provider can apply, optionally, the Veridas anti-spoofing detection services to determine that the captured selfie photo is not a photo of a screen (not available for web capture technology) and to check that the selfie photo and the smiling photo corresponds to the same person. This functionality requires the use of the das-Face API.

Finally, the service provided calls the das-FaceQR API with the selfie photo captured by the user. Moreover, the localization and the concert date are included as contextual data, as well as the access door and the seat assigned to the user. With this information, das-FaceQR generates a biometric QR.



The previous image represents an example of version 23 and M redundancy QR, which is adequate for storing the biometric information and the contextual data associated to the concert.

The user receives an email confirming that the purchase process has ended (the flow related to the payment process is obvious in the description) and that the ticket is available. The user has two options, print the entry with the biometric QR or save it in its wallet.

Access to the concert

The concert's day, the user goes to the place where it takes place.

On the day of the concert, the user goes to the facilities where it takes place. A biometric door is installed in the access control. This door must contain the following elements.

- A camera for capturing the user image.
- A QR code reader, although they can optionally be read through an image processing software from the previous camera.

The user approaches the camera and shows his biometric QR to the reader. The displayed QR can be represented on the paper entry or on the wallet of the user's mobile device. In this way, the authentication process begins with something that the user is (his face) in combination with something that the user has (his physical or digital biometric QR).

In any case, the software installed on the door sends the facial image captured with the biometric door camera and the biometric QR to das-FaceQR to verify that the user has the right to access (it is obvious at this point if das-FaceQR is installed on-premises or is served from the cloud).

das-FaceQR receives the above information. First, it checks the integrity of the information obtained by signing the biometric QR, to ensure that there is no alteration on the biometric QR. If the integrity of the information can be assured, das-FaceQR decrypts the biometric vector. Following this process, das-FaceQR sends das-Face the registration biometric vector and the recently captured image at the concert access door. das-Face reads the biometric model version that has to be used for the biometric comparison and executes the corresponding comparison. Finally, a similarity value and the contextual data read is returned to the service provider.

The service provider then executes business logic. For example, it checks the contextual data information regarding the concert date and the gateway. If they are correct, it goes on to check the business logic based on the biometrics score. For example, if the comparison score obtained by das-FaceQR is greater than 90%, it authorizes the user access.

Finally, after the authorization, the access door opens and the user accesses the concert facility.

FAQs

The ticket buyer illegally resells its ticket. Can the illegitimate buyer access the concert?
It cannot access the concert. The biometric comparison between the photo captured at the access door and the biometric QR would not be satisfactory, obtaining a low similarity score.

The ticket buyer loses his ticket. Can a person who finds it access the concert?
You cannot access the concert. As in the previous case, the biometric comparison between the photo captured at the access door and the biometric QR would not be satisfactory, obtaining a low similarity score.

The buyer of the ticket cannot attend the concert and decides to return the ticket or sell it by the legal means established for this purpose. How can it be articulated?
The service provider should establish an inbound return or delegation process for the ticket. In other words, a process that allows the already purchased ticket to be authenticated in order to return it or transfer it to another person. This authentication process would consist on validating the ownership of the ticket by using a selfie photo and the biometric QR of the entrance, thus allowing the return of the money or the transfer to another third person. This third person would carry out a process of generating their biometric QR following the usual process.

The concert is repeated on Friday and Saturday. The user bought the ticket for Friday. Can the user also access the concert on Saturday?
You cannot access the concert, as long as the service provider establishes a validity date for the biometric QR and verifies it after reading the contextual data by das-FaceQR

The user gets the wrong access door to the concert facility. Can the user access the facility?
As in the previous case, you cannot access the concert, as long as the service provider sets the

value of the access door in the contextual data and checks it after reading by das-FaceQR.

The ticket buyer manipulates the data of the seat number in the biometric QR to access a privileged area. Can the user access the concert?
The user cannot access the concert. The integrity check carried out by das-FaceQR verifies that the stored data, both the biometric credential and the contextual data have not been modified. This is done thanks to the signing of the data.

The user makes a fake biometric QR to access the concert. To do this, he uses a biometric credential that he previously used to travel by train and adds the contextual information associated with the concert. Can the user access the concert?
The user cannot access the concert. The integrity check carried out by das-FaceQR verifies that the stored data, both the biometric credential and the contextual data have not been modified. This is done thanks to the signing of the data. Additionally, the encryption of the biometric vector will be done with a different key for each client in future versions of das-FaceQR. This implies that decryption of the biometric vector with the password of the company which organizes the concert would not be possible, since the biometric vector was originally encrypted with the password of the railway transport company.

Annex C: Changelog History

dasFaceQR 2021Q1

2.1. Added and improved

- New endpoints which allow to generate standard QRs and verify them
- Automatic detection New endpoints which allow to generate standard QRs and verify them
-
- Automatic detection of QR encoding (parameter “isDecoded” is no longer needed)New endpoints which allow to generate standard QRs and verify them
-
- Automatic detection of QR encoding (parameter “isDecoded” is no longer needed)of QR encoding (parameter “isDecoded” is no longer needed)

dasFaceQR 2020Q4

2.1. Added

- Added support for Aztec Codes.
- New parameter “isDecoded” indicates if the QR code data has been decoded on the client side or not.

2.2. Fixed

- Some errors that used to return 500 codes now return 400.

dasFaceQR 2020Q2

2.1. Added

- The selfie image is now optional. It’s possible to generate a biometric credential containing only a vector, only contextual data, or both.
- It’s possible to select which of the three supported algorithms will be used to sign a QR code. The endpoint has been extended such that the algorithm can be specified: /credential/qr-code/algorithm/<algo-name>.
- Now additional custom data can be added at the beginning of the QR code. This data will not be part of the signature.

2.2. Fixed

- Various fixes in the generated Passbook files. Some readers were unable to read them.
- QR length capped below maximum possible size
- Under some circumstances the signature algorithm could not be selected

2.3. Deprecated

- As announced in previous release GET /size-limits endpoint will not be available any longer.

dasFaceQR 2019Q4

2.1. Added

- New endpoint `/credential/qr-code/passbook/event` to generate Apple Passbook files.
- Newly-generated QR codes will ship a biometric vector with a reduced BPC (bits per cluster) of 8, in order to generate smaller QR codes.
- New endpoint `/models`
- Support for signature algorithm ISO/IEC 9796-2.