UNINETT

Multifactor authentication

Deployment guide

Document History

Version	Date	Author	Comments
1.4	Feb 2017	Jaime Pérez	Fix a couple of typos in Appendix 1, norEduPersonServiceAuthnLevel examples.
1.3	Oct 2015	Jaime Pérez	Update for final settings in production.
1.2	Jul 2015	Hildegunn Vada	Removed URLs for Feide Authenticator API until further.
1.1	Jan 2015	Jaime Pérez	Update with new attributes and formats from norEduPerson.
1.0	Dec 2014	Jaime Pérez	First version of this document.

UNINETT AS

Abels gate 5 – Teknobyen
P.O. Box: NO-7465 Trondheim
Sør-Trøndelag, Norway

+47 73 55 79 00
support@feide.no

Introduction

Feide, the Identity Federation of the Norwegian National Research and Education Network (UNINETT), is introducing the possibility to use a second factor of authentication as a requisite to complete the login process and minimise the possibility of identity theft, especially for those services that require higher guarantees on the security of the whole system.

This document describes the technical details and requisites involved in the deployment of a multifactor authentication solution in Feide, as well as the rationale behind them. Institutions willing to deploy this solution should follow this document as a technical guide for an interoperable and secure implementation. This guide, however, does not pretend to be exhaustive or to provide specific instructions for every possible setup, or to describe the administrative procedures that every institution should have in place for a successful and secure deployment.

Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119].

The use of SHOULD, SHOULD NOT, and RECOMMENDED reflects broad consensus on deployment practices intended to foster both interoperability and guarantees of security and confidentiality needed to satisfy the requirements of many organizations that engage in the use of federated identity. Deviating may limit a deployment's ability to technically interoperate without additional negotiation, and should be undertaken with caution.

Motivation

The main purpose of a multifactor authentication solution is to verify the identity of the user by other means apart from the traditional username and password authentication. Second factors are usually something that the user physically has, commonly combined with something the user knows (like a token with a PIN code that generates one-time passwords).

In order to use most of the multifactor solutions available in the market, some information about the other factors needs to be stored. Since Feide does not store information about the end users in any way, the institutions need to store themselves the relevant information for their own users and the other factors they want to support.

Feide has so far implemented support for two different multifactor authentication mechanisms: SMS-based and time-based one-time passwords. Here we discuss their basic characteristics as well as their main challenges.

SMS-based one-time password

Authentication based on text messages is probably one of the simplest and most widespread multifactor mechanisms. When a user has identified with a username and password, a random code is sent to the mobile phone number associated with that account, and the user needs to enter it to continue. That way we can verify that the user is in possession of a specific device he or she is supposed to own.

In the case of Feide, many institutions are already registering and storing the mobile phone numbers of their users. However, experience shows that those numbers are not strictly validated, and they may often contain errors, typos or simply be invalid, either due to automated provisioning or the users manually entering them. Besides, it is also desirable that the institutions are able to signal whether a phone number should be used or not for multifactor authentication. This means the phone numbers currently registered at every institution cannot be used right away, so they must be provisioned again, verifying their authenticity and format.

Time-based one-time password

Feide has implemented support for a *time-based one-time password* (TOTP) authentication method that is compatible with [RFC 6238]. It is more specific though, as it uses a subset of the mechanisms described in the standard and restricts how others are used. This authentication method (referred to as *Authenticator* from now on) can be used with hardware tokens, but also with any device capable of running software implementing this method, which is part of its success.

Most typically, the user configures a smartphone with an *Authenticator* application by manually entering a shared secret on the application or by scanning a QR code. Later on, the device can be used to generate the six-digit codes used for authentication. Note that even though this mechanism was originally implemented by Google in its *Google Authenticator* application, there is also a wide range of clients that support this mechanism.

The *time-based one-time password* authentication mechanism is based on a secret shared between a device owned by the end user (a mobile phone, a hardware token or some software running on a laptop) and the authenticating party. In this case the authenticating party is Feide, as multifactor authentication happens right after traditional username and password authentication, but Feide by design does not store any authentication data for the users. This means the secret shared between Feide and the user must be stored in the directory of the home institution of the user, the same way as passwords.

This shared secret is necessary to setup and use an *Authenticator* application to authenticate a user. In a way, it is equivalent to a password. This means we have to handle it with the same security considerations as we would handle a password. Particularly, if the shared secret is stored in plain in the user's entry in the directory, anybody able to read the entries of other users would be able to setup an *Authenticator* instance identifying any particular user, completely defeating the purpose of multifactor authentication.

In order to avoid the *Authenticator* secrets to be widely readable for anybody with access to the directory, they must be protected by some mechanism so that only the authenticating party (Feide in this case) can use them. This is achieved by using public key cryptography to encrypt the secrets, so that anybody can encrypt, but only Feide can

decrypt them. This is different to usual password protection mechanisms, as passwords are commonly ***hashed*** with a ***one-way*** function instead of encrypted. This method cannot be used with the ***Authenticator*** secrets since Feide needs the original value to generate the one-time password, and the only place where it can get that value from is the user directory.

Technical specification

All the information related to the usage of multifactor authentication by a certain user must be stored directly in the user's entry at the institutional directory. In order to accommodate such information without interfering with other attributes already in the directories, and to guarantee the technical and administrative requisites needed, Feide has extended the ***norEduPerson*** schema [norEdu] with two additional attributes:

- *norEduPersonServiceAuthnLevel*: a multivalued attribute that lets the institution decide whether multifactor authentication is enabled or not for the current user, the strength of the authentication mechanism, and for which services.
- *norEduPersonAuthnMethod*: a multivalued attribute that lets the institution specify which multifactor methods can be used by the current user.

Enforcing multifactor authentication

In order to allow someone to use multifactor authentication in Feide, one or more multifactor methods MUST be defined. However, this requirement is not sufficient, as multifactor authentication MUST also be enabled for the service being accessed or for a specific user. Institutions can configure which services should use multifactor authentication in Feide's Customer Portal[1].

Alternatively, the general configuration can be overridden on a per-user basis by adding the ***norEduPersonServiceAuthnLevel*** attribute of the [norEdu] schema to the specific

[1] See https://kunde.feide.no/.

subject. The attribute consists of two strings separated by a white space. The first one is the fixed string `urn:mace:feide.no:spid:` followed by one of these values:

- The keyword `all`, used to enforce multifactor authentication for all services.
- A number identifying a specific service, where this number corresponds to the *Service ID* that can be found in Feide's Customer Portal for that service.

These values MUST be followed by another string representing the assurance level desired for either all or the particular services specified. Currently, Feide supports only the third level described by the Norwegian *Framework for authentication and non-repudiation in electronic communications with and in the public sector* [FAD08]. The level of assurance can be expressed with the string `urn:mace:feide.no:auth:level:fad08:` followed by the numerical value identifying the level, in this case, 3.

Please note that none of the authentication methods currently supported by Feide provide level 4 according to this framework. Therefore, requiring that level would make it impossible for users to access the service or services affected.

SMS authentication

Institutions can enable multifactor authentication based on text messages by enabling multifactor authentication as described in the previous section, and adding a *norEduPersonAuthnMethod* attribute to every user who SHOULD be able to use this mechanism. This attribute is multivalued and each value MUST contain:

- The fixed string `urn:mace:feide.no:auth:method:sms` identifying the SMS multifactor method in Feide.
- The mobile phone number associated with the user in international format, including the country code prefixed with the plus sign (+). Phone numbers MUST NOT have blank spaces, dashes or any other character different than numbers, with the only exception of the aforementioned international code prefix.

Optionally, a label identifying the device associated with the phone number can be added after these two mandatory elements. If no label is specified, the last digits of the phone number are displayed. Labels take the form `label=value`, where `value` is a

string that we want to be displayed in order to identify the associated number. The value MUST be URL-encoded as defined by [RFC3986] and therefore comply with the following rules:

- Equal signs (=) MUST be escaped according to percent encoding[2] as defined by [RFC 3986], that is, substituted by the string `%3D`.
- Spaces MUST be escaped according to percent encoding as defined by [RFC 3986], that is, substituted by the string `%20`.
- Percent signs (%) MUST be escaped according to percent encoding as defined by [RFC 3986], that is, substituted by the string % 25.
- No single or double quotes are allowed surrounding the values.

All parts of the *norEduPersonAuthnMethod* attribute MUST be separated by a single blank space and keep the order specified here, such as the resulting value has the following format:

```
urn:mace:feide.no:auth:method:sms +<COUNTRY_CODE><NUMBER> label=<A_LABEL>
```

Refer to Appendix 1 for examples on values that are correctly formatted and values that are not.

Authenticator

In this section we present the technical details for storing and encrypting secrets in user directories, so that they can be used securely to perform multifactor authentication with the *Authenticator* method.

Authenticator secrets consist of 16 base32-encoded characters. This means that valid secrets can only contain:

- Uppercase ASCII letters from A to Z, both included.
- Digits from 2 to 7, both included.

There is no restriction on the proportion of letters or digits, though, so secrets can be random strings base32-encoded with the aforementioned fixed length of 16 characters.

[2] See http://en.wikipedia.org/wiki/Percent-encoding

It is therefore RECOMMENDED to randomly generate these secrets. Institutions MUST NOT use functions to derive the secret from known or guessable values like the user identifier or e-mail address. If the *Authenticator* secret can be derived from other information related to the user, it is then useless as anybody could guess the right secret for a specific person.

Random secret generation

For your convenience, Feide provides a simple web service for the generation of random *Authenticator* secrets. This web service accepts no parameters, and returns a JSON structure containing the random value. Contact Feide support in order to get more information about it.

This web service is rate limited, so if more than a fixed amount of requests are made during a period of time, access will be denied to it. Those willing to use this web service to generate random secrets for their users in automated, bulk operations, SHOULD ask Feide for an authorization token that they will be able to use with a less strict rate limiting policy.

The response obtained from the web service will be a *JSON* encoded string with the form:

```
{"random": "RANDOM_SECRET"}
```

where `RANDOM_SECRET` is a randomly generated 16-character string, base32 encoded.

Make sure to check the return status of the responses, which will be `200 OK` in case of success, and the `Content-Type` response header, which MUST contain the `application/json` *MIME* type. If the response does not fulfil any of both requisites, users MUST NOT expect the contents in the response to be a valid *JSON* structure.

Automated secret encryption

Feide provides also a web service for the encryption of *Authenticator* secrets, so that only Feide can decrypt them and they can be safely stored in an institutional user directory. Contact Feide support in order to get more information about it.

As in the previous case, this web service is rate limited, so if more than a fixed amount of requests are made during a period of time, access will be denied to it. Those willing to

use this web service to encrypt secrets for their users in automated, bulk operations, SHOULD ask Feide for an authorization token that they will be able to use with a less strict rate limiting policy.

All requests to this web service MUST use the *HTTP POST* method,[3] and only one parameter named *secret* is accepted, containing a serialized *JSON* structure in the following form:

```
{"secret": "SECRET_TO_ENCRYPT"}
```

where `SECRET_TO_ENCRYPT` contains the 16-character, base32-encoded shared secret. The web service will encrypt the secret provided with Feide's public key, and produce the following *JSON* structure as output:

```
{"ciphertext": "ENCRYPTED_SECRET"}
```

where `ENCRYPTED_SECRET` is the value that can be used in the user's entry in the institutional directory.

Manual secret encryption

Those willing to integrate *Authenticator* secret encryption with their own processes and systems can do it without depending on the API provided by Feide. Secrets are encrypted using the *JSON Web Encryption* [JWE] specification with a public key specific for this use and published in the Feide website[4]. The secret used as input for the encryption algorithm MUST be encoded as a *JSON* object, with the form:

```
{"secret": "SECRET_TO_ENCRYPT"}
```

where `SECRET_TO_ENCRYPT` is the randomly generated secret that has been configured on the user's device. The [JWA] specification defines multiple algorithms for key and content encryption. Currently, Feide supports only the `RSA-OAEP` key management algorithm and the `A128CBC_HS256` content encryption algorithm. Appendix 2 presents code examples on how to encrypt the *Authenticator* secrets suitable for their use in Feide in different programming languages.

[3] Make sure to make all requests to this endpoint protected with HTTPS, as failing to do so could leak the secret that you want to encrypt.
[4] See https://metadata.feide.no/.

Storage of secrets

Secrets for *Authenticator* instances MUST be stored in the user's entry in the corporate directory. The secrets themselves will be stored in the *norEduPersonAuthnMethod* attribute defined by the [norEdu] schema. The values of this attribute MUST contain:

- The fixed string `urn:mace:feide.no:auth:method:ga` identifying the *Authenticator* multifactor method in Feide.
- The Authenticator secret encrypted with Feide's public key as described in the previous section.

Optionally, a label identifying a specific instance of the *Authenticator* can be included. Labels are prefixed by the string `label=`, which is omitted if no label is attached to this *Authenticator* instance. All parts of the attribute value MUST be separated by a single blank space and keep the order, such as the resulting value has the following format:

```
urn:mace:feide.no:auth:method:ga <ENCRYPTED_SECRET> label=<A_LABEL>
```

Both the encrypted secret and the label (excluding its prefix) MUST comply with the following rules:

- Equal signs (=) MUST be escaped according to percent encoding[5] as defined by [RFC 3986], that is, substituted by the string `%3D`.
- Spaces MUST be escaped according to percent encoding as defined by [RFC 3986], that is, substituted by the string `%20`.
- Percent signs (%) MUST be escaped according to percent encoding as defined by [RFC 3986], that is, substituted by the string % 25.
- No single or double quotes are allowed surrounding the values.

API authorization

In order to obtain a more flexible rate limiting policy and use Feide's APIs with automated, bulk processes, you SHOULD ask the Feide team for an authorization token. Once a token has been granted, there are two different ways to use it with Feide's API:

- Include them as an *HTTP GET* or *POST* parameter named `access_token`.

[5] See http://en.wikipedia.org/wiki/Percent-encoding

- Include them inside an *HTTP header* named *Authorization* with the following structure: `Authorization: Bearer <API_TOKEN>`, where `<API_TOKEN>` is the authorization token granted by Feide.

References

[FAD08] *Rammeverk for autentisering og uavviselighet i elektronisk kommunikasjon med og I offentlig sektor*, April 2008;
https://www.regjeringen.no/globalassets/upload/fad/vedlegg/ikt-politikk/eid_rammeverk_trykk.pdf

[FeideTech] *Feide Technical Guide*, Sep. 2014;
https://www.feide.no/sites/feide.no/files/documents/Feide_technical_guide.pdf

[JWA] M.Jones et al, *JSON Web Algorithms (JWA)*, Jan. 2015;
https://datatracker.ietf.org/doc/draft-ietf-jose-json-web-algorithms/

[JWE] M. Jones et al, *JSON Web Encryption (JWE)*, Jan. 2015;
https://datatracker.ietf.org/doc/draft-ietf-jose-json-web-encryption/

[norEdu] H. Vada, J. Pérez, *norEdu* Object Class Specification*, Sep. 2015;
https://www.feide.no/feide/sites/drupal.uninett.no.feide/files/documents/norEdu_spec.pdf

[RFC 2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119, March 1997;
https://www.rfc-editor.org/rfc/rfc2119.txt

[RFC 3986] T. Berners-Lee et al, *Uniform Resource Identifier (URI): Generic Syntax*, Jan. 2005, section 2.4;
https://tools.ietf.org/html/3986

[RFC 6238] D. M'Raihi et al, *TOTP: Time-Based One-Time Password Algorithm*, May 2011;
https://tools.ietf.org/html/rfc6238

Appendix 1: examples

The following are valid phone numbers to be used with SMS authentication:

- `+4701234567`
- `+34012345678`
- `+10123456789`

The following are NOT valid phone numbers to be used with SMS authentication:

- `004701234567` (it uses the `00` prefix instead of +)
- `4701234567` (it does not use the international prefix +)
- `+470123456` (the number has incorrect length)
- `+47 01 23 45 67` (it contains spaces)
- `+1-012-345-6789` (it contains dashes)

The following are valid secrets to be used with the *Authenticator* method:

- `ABCDEFGHIJ234567`
- `MKMPIDBZ2UOUSCTZ`
- `ABCDEFGHIJKLMNOP`
- `2345672345672345`

The following are NOT valid secrets to be used with the *Authenticator* method:

- `abcdefghijklmnop` (it uses lowercase characters)
- `0123456789012345` (it uses numbers other than those from 2 to 7)
- `0123456789` (it uses invalid numbers and is shorter than 16)
- `234567ABC` (it uses valid characters and numbers but is shorter than 16)
- `ABCDEFGHIJKLMNOPQRSTUVWXYZ` (it users valid characters but is longer than 16)
- `ABC +=1234567DEF` (it uses invalid symbols)

The following are valid values for the *norEduPersonServiceAuthnLevel* attribute. Please note that line feeds are used for display purposes and should be disregarded:

- `urn:mace:feide.no:spid:all urn:mace:feide.no:auth:level:fad08:3` (enable multifactor authentication for all services)
- `urn:mace:feide.no:spid:123 urn:mace:feide.no:auth:level:fad08:3` (enable multifactor authentication for the service with *Service ID* number 123)

The following are valid values for the *norEduPersonAuthnMethod* attribute. Please note that line feeds are used for display purposes and should be disregarded.

- `urn:mace:feide.no:auth:method:ga eyJhbGciOiAiUlNBLU9BRVAiLCAiZW 5jIjogIkExMjhDQkMtSFMyNTYifQ.X7IU3zolmVtGzXxfKIxJLyvP5KNnqEdDG JBQNx8Y8VwvvdoeGjTiiU0V5OJKykylEhUITVTQ1l5snlBndVtVSjlkhK7CVZx 12OUcferIIC90tBg-GJRbom-RWVIYbXdB1jcUwMaUPZB49yoquhP1mTfFv76e9 5uize124XfyowcrM6dnPWhSSuPgDzp3_oA8e5Z6U1qzm-mDHe3BF1krNBXQjwx HWY4lC1zd7wbIGhBcngqmK8-ebRyDelMUpbOSgADWiQxdTeEkXVSOEn3JKiRuo YhggePNWM1rGnarooUktnuxdK6pggRSIAPkzM-ghJEDPtuk5gc5NSWNBE0x3g. LgmSqnSduW8WnpUjPfF4Gg.In2Wd2AU-6OMRxFily8EbKtmG4gC5EnKyyYZthk aM9w.MWE29ywstyWKCrCUttUYZg label=Mobile`

  (for a device labelled "*Mobile*")

- `urn:mace:feide.no:auth:method:ga eyJhbGciOiAiUlNBLU9BRVAiLCAiZW 5jIjogIkExMjhDQkMtSFMyNTYifQ.X7IU3zolmVtGzXxfKIxJLyvP5KNnqEdDG JBQNx8Y8VwvvdoeGjTiiU0V5OJKykylEhUITVTQ1l5snlBndVtVSjlkhK7CVZx 12OUcferIIC90tBg-GJRbom-RWVIYbXdB1jcUwMaUPZB49yoquhP1mTfFv76e9 5uize124XfyowcrM6dnPWhSSuPgDzp3_oA8e5Z6U1qzm-mDHe3BF1krNBXQjwx HWY4lC1zd7wbIGhBcngqmK8-ebRyDelMUpbOSgADWiQxdTeEkXVSOEn3JKiRuo YhggePNWM1rGnarooUktnuxdK6pggRSIAPkzM-ghJEDPtuk5gc5NSWNBE0x3g. LgmSqnSduW8WnpUjPfF4Gg.In2Wd2AU-6OMRxFily8EbKtmG4gC5EnKyyYZthk aM9w.MWE29ywstyWKCrCUttUYZg label=My%20mobile%20phone`

  (for a device labelled "*My mobile phone*")

- `urn:mace:feide.no:auth:method:ga eyJhbGciOiAiUlNBLU9BRVAiLCAiZ W5jIjogIkExMjhDQkMtSFMyNTYifQ.X7IU3zolmVtGzXxfKIxJLyvP5KNnqEdD GJBQNx8Y8VwvvdoeGjTiiU0V5OJKykylEhUITVTQ1l5snlBndVtVSjlkhK7CVZ x12OUcferIIC90tBg-GJRbom-RWVIYbXdB1jcUwMaUPZB49yoquhP1mTfFv76e 95uize124XfyowcrM6dnPWhSSuPgDzp3_oA8e5Z6U1qzm-mDHe3BF1krNBXQjw xHWY4lC1zd7wbIGhBcngqmK8-ebRyDelMUpbOSgADWiQxdTeEkXVSOEn3JKiRu oYhggePNWM1rGnarooUktnuxdK6pggRSIAPkzM-ghJEDPtuk5gc5NSWNBE0x3g .LgmSqnSduW8WnpUjPfF4Gg.In2Wd2AU-6OMRxFily8EbKtmG4gC5EnKyyYZth kaM9w.MWE29ywstyWKCrCUttUYZg label=%25%20%3D%20%25`

(for a device labelled "% = %")

The following are NOT valid values for the *norEduPersonAuthnMethod* attribute. Please note that line feeds are used for display purposes and should be disregarded.

- `urn:mace:feide.no:auth:method:authenticator eyJhbGciOiAiUlNBLU9BRVAiLCAiZW5jIjogIkExMjhDQkMtSFMyNTYifQ.X7IU3zolmVtGzXxfKIxJLyvP5KNnqEdDGJBQNx8Y8VwvvdoeGjTiiU0V5OJKykylEhUITVTQ1l5snlBndVtVSjlkhK7CVZx12OUcferIIC90tBg-GJRbom-RWVIYbXdB1jcUwMaUPZB49yoquhP1mTfFv76e95uize124XfyowcrM6dnPWhSSuPgDzp3_oA8e5Z6U1qzm-mDHe3BF1krNBXQjwxHWY4lC1zd7wbIGhBcngqmK8-ebRyDelMUpbOSgADWiQxdTeEkXVSOEn3JKiRuoYhggePNWM1rGnarooUktnuxdK6pggRSIAPkzM-ghJEDPtuk5gc5NSWNBE0x3g.LgmSqnSduW8WnpUjPfF4Gg.In2Wd2AU-6OMRxFily8EbKtmG4gC5EnKyyYZthkaM9w.MWE29ywstyWKCrCUttUYZg label=Mobile`

  (the method identifier URN does not correspond with Feide's Authenticator method)

- `urn:mace:feide.no:auth:method:ga eyJhbGciOiAiUlNBLU9BRVAiLCAiZW5jIjogIkExMjhDQkMtSFMyNTYifQ..MWE29ywstyWKCrCUttUYZg label=Mobile`

  (the encrypted secret is not a valid secret)

- `urn:mace:feide.no:auth:method:ga ABCDEFGHIJ234567 label=Mobile`

  (the Authenticator secret is not encrypted)

- `urn:mace:feide.no:auth:method:ga eyJhbGciOiAiUlNBLU9BRVAiLCAiZW5jIjogIkExMjhDQkMtSFMyNTYifQ.X7IU3zolmVtGzXxfKIxJLyvP5KNnqEdDGJBQNx8Y8VwvvdoeGjTiiU0V5OJKykylEhUITVTQ1l5snlBndVtVSjlkhK7CVZx12OUcferIIC90tBg-GJRbom-RWVIYbXdB1jcUwMaUPZB49yoquhP1mTfFv76e95uize124XfyowcrM6dnPWhSSuPgDzp3_oA8e5Z6U1qzm-mDHe3BF1krNBXQjwxHWY4lC1zd7wbIGhBcngqmK8-ebRyDelMUpbOSgADWiQxdTeEkXVSOEn3JKiRuoYhggePNWM1rGnarooUktnuxdK6pggRSIAPkzM-ghJEDPtuk5gc5NSWNBE0x3g.LgmSqnSduW8WnpUjPfF4Gg.In2Wd2AU-6OMRxFily8EbKtmG4gC5EnKyyYZthkaM9w.MWE29ywstyWKCrCUttUYZg=== label=Mobile`

  (the encrypted secret contains equal signs that are not percent encoded)

- `urn:mace:feide.no:auth:method:ga eyJhbGciOiAiUlNBLU9BRVAiLCAiZW5jIjogIkExMjhDQkMtSFMyNTYifQ.X7IU3zolmVtGzXxfKIxJLyvP5KNnqEdDGJBQNx8Y8VwvvdoeGjTiiU0V5OJKykylEhUITVTQ1l5snlBndVtVSjlkhK7CVZx12OUcferIIC90tBg-GJRbom-RWVIYbXdB1jcUwMaUPZB49yoquhP1mTfFv76e95uize124XfyowcrM6dnPWhSSuPgDzp3_oA8e5Z6U1qzm-mDHe3BF1krNBXQjwxHWY4lC1zd7wbIGhBcngqmK8-ebRyDelMUpbOSgADWiQxdTeEkXVSOEn3JKiRuoYhggePNWM1rGnarooUktnuxdK6pggRSIAPkzM-ghJEDPtuk5gc5NSWNBE0x3g.LgmSqnSduW8WnpUjPfF4Gg.In2Wd2AU-6OMRxFily8EbKtmG4gC5EnKyyYZth`

```
kaM9w.MWE29ywstyWKCrCUttUYZg label=My mobile %phone
```

(the label contains spaces and percent signs that are not percent encoded)

The following is a valid QR code to configure an *Authenticator* instance with the label `My device` and the secret `ABCDEFGHIJ234567`:

My device

Appendix 2: code examples

Authenticator secret codes can be easily encrypted for their use in Feide by means of several libraries providing *JSON Web Encryption* for different programming languages. Here we provide a couple of examples for your convenience.

PHP

The following example encrypts an Authenticator secret with help of the `gree/jose` library[6]. It assumes *composer* is used to install the library, and an autoloader is available in the `vendor` directory at the same location of the PHP script.

[6] See https://github.com/gree/jose.

```php
<?php
include("vendor/autoload.php");

// JSON-encode the Authenticator secret
$plaintext  = array("secret" => "<AUTHENTICATOR_SECRET");
// initialize JWE object
$jwe = new JOSE_JWE(json_encode($plaintext));
// encrypt with Feide's key and proper algorithms
$jwe->encrypt(
    '<FEIDE_AUTHENTICATOR_PUBLIC_KEY',
    'RSA-OAEP',
    'A128CBC-HS256'
);
// print the result
echo $jwe->toString();
```

Python

This example uses the `bifurcation/pyjose` library[7] to encrypt a *Authenticator* secret. It assumes the library is correctly installed and can be imported from python.

```python
#!/usr/bin/env python
from Crypto.PublicKey import RSA
import jose
import jose.josecrypto
from jose.serialize import serialize_compact
import json

# prepare public key for use
pub_key = "<FEIDE_AUTHENTICATOR_PUBLIC_KEY>"
pub_key = pub_key.strip()
pub_key = RSA.importKey(pub_key)
key = josecrypto.exportKey(pub_key, "RSA")

# prepare protected header (select algorithms)
jwe_header = { "alg": "RSA-OAEP", "enc": "A128CBC-HS256" }

# encrypt the plaintext
plaintext = json.dumps({ "secret": "<AUTHENTICATOR_SECRET>"})
jwe = jose.encrypt(jwe_header, [key], plaintext, protect="*")

# print compact serialization of the JWE
print serialize_compact(jwe)
```

[7] See https://github.com/bifurcation/pyjose.

The `Demonware/jose` library[8] can be used alternatively, provided that the version installed is at least `1.0.0`. The following example demonstrates its use to generate *Authenticator* secrets.

```
#!/usr/bin/env python
import jose

# prepare public key for use
pub_key = "<FEIDE_AUTHENTICATOR_PUBLIC_KEY>"
pub_key = pub_key.strip()

# configure algorithms
alg = "RSA-OAEP"
enc = "A128CBC-HS256"

# calculate jwe header
header = jose.b64encode_url(jose.json_encode({
    "enc": enc,
    "alg": alg,
    jose._TEMP_VER_KEY: jose._TEMP_VER
}))

# encrypt the plaintext
plaintext = "<AUTHENTICATOR_SECRET>"
claim = {"secret": plaintext.strip()}
jwe = jose.encrypt(
    claim,
    {"k": pub_key},
    alg=alg,
    enc=enc,
    adata=header
)

# export compact serialization of the JWE
print jose.serialize_compact(jwe)
```

[8] See https://github.com/Demonware/jose.