eIDAS Constructed Attributes Specification for the Swedish eID Framework

Version 1.1 - 2019-08-30 - *Draft version*

2019-315

> *Previous registration number: ELN-0611*

Table of Contents

1. Introduction

This document extends “Attribute Specification for the Swedish eID Framework”, [EidAttributes], providing specifications for constructed attributes.

The concept of constructed attributes is introduced in Swedish national authentication nodes (proxy nodes) delivering identity assertions to Swedish Service Providers based on user authentication with a foreign eID.

A constructed attribute is an attribute that was not delivered by the foreign Identity Provider service, but was constructed in the Swedish authentication node by applying defined rules and algorithms to the authenticated user (subject) received from the foreign Identity Provider service (typically an eIDAS node).

1.1. Requirement key words

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” are to be interpreted as described in [RFC2119].

These keywords are capitalized when used to unambiguously specify requirements over protocol features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

2. Provisional Identifier

The Attribute Specification for the Swedish eID Framework, [EidAttributes], defines the attributes `prid` and `pridPersistence`.

The `prid` attribute holds a unique identifier for a person derived from attributes provided from another country. The purpose of this attribute is to provide a common unique attribute for an authenticated user independently of the attribute set or the characteristics of these attributes provided by the authentication service in the other country.

The `pridPersistence` attribute provides an indicator of the expected persistence of the `prid` identifier over time. The value in this attribute is determined by assessing the persistence of underlying foreign attributes from a particular source used in a particular `prid` generation algorithm.

This document defines a set of `prid` algorithms, when to use each algorithm and the resulting `pridPersistence` value.

2.1. Provisional Identifier (prid) Attribute

The provisional identifier (`prid`) attribute is a SAML attribute identified by the SAML attribute name `urn:oid:1.2.752.201.3.4`.

The `prid` attribute holds a string value containing the following data:

> **{2 letter ISO 3166 country code of eID country}** + **":"** + **{10..30 character identifier}**

Syntactically, provisional ID is defined by the following regular expression:

```
^[A-Z]{2}:[0-9a-z][0-9a-z-]{8,28}[0-9a-z]$
```

Examples:

> NO:29078534891
>
> DK:09208-2002-2-194967071622

The country code is the 2 letter ISO 3166 country code expressed in upper case letters, for example, “SE” for Sweden and “NO” for Norway. This identifies the country that issued the eID used to authenticate the user (i.e., provided the infrastructure to identify the person). This is not necessarily the person’s actual citizenship or country of residence.

The identifier component holds a minimum of 10 and a maximum of 30 characters. The primary reason for this is to provide an identifier that can be displayed to a user and still relatively convenient to write down or communicate by humans in case of problems. The characters in the identifier component are restricted to the numeric characters 0-9, the letters a-z and the hyphen character “-“ (0x2D) serving as delimiter. Letters “a-z” MUST be lower case. Should a provisional ID ever be presented with upper case letters then such letter should be matched using case insensitive matching (e.g. “a” is equivalent to “A”). The identifier component MUST NOT start or end with a hyphen character. The resulting ID MUST have at least 8 characters that are not a hyphen character, for example, the character sequence “1-2-3-4-56” is not allowed as it only holds 6 distinguishing ID characters.

2.2. Provisional Identifier Persistence Indicator (pridPersistence) Attribute

The provisional identifier (`pridPersistence`) attribute is a SAML attribute identified by the SAML attribute name `urn:oid:1.2.752.201.3.5`.

The pridPersistence attribute holds a string value containing the following data:

> **{1 letter Identifier (A, B or C)}**

Examples:

> A
>
> B
>
> C

Value definitions:

Value	Defined meaning
A	Persistence over time is expected to be comparable or better than a Swedish personal identity number (personnummer). This means that the identifier typically is stable throughout the lifetime of the subject and is typically preserved even if the subject changes address, name or civil status.
B	Persistence over time is expected to be relatively stable, but lower than a Swedish personal identity number (personnummer). This means that the identifier typically remains unchanged as long as the person does not change address, name or civil status. Such or similar event may cause the identifier to change but the identifier will not change just because the subject gets a new eID (electronic identification means) or changes eID provider.
C	No expectations regarding persistence over time. The identifier may change if the subject changes eID or eID provider.

2.3. Algorithms

This section defines algorithms for generating the identifier component of `prid` attribute values. The identifier component makes up the characters following the ":" (colon) character in the `prid`.

2.3.1. Algorithm: default-eIDAS

Name: `default-eIDAS`

Input values:

- `eidasID` - The identifier string value from the eIDAS PersonIdentifier attribute from the attribute source (identified by the attribute name `http://eidas.europa.eu/attributes/naturalperson/PersonIdentifier`.

Calculated values:

- `strippedID` = `eidasID` after:
  i. removing the 6 leading characters matching the regular expression `^[A-Za-z]{2}[\/](SE\|se)[\/]` (e.g., "NO/SE/"), and;
  ii. removing any white space and non-printable characters.
- `normalizedID` = `strippedID` converted according to the following steps:
  i. converting all upper case letters [A-Z] to lower case, and,
  ii. replacing with a single "-" character, all sequences of characters of length (1…n) that does not contain [0-9] or [a-z], and,
  iii. remove any leading or trailing "-" characters.

Result:

If length of `normalizedID` < 10 characters:

> Return `normalizedID` padded with leading “0” (zero) characters until length = 10 characters.

If length of `normalizedID` 10 - 30 characters:

> Return `normalizedID`

If length of `normalizedID` > 30 characters**:**

> Return the string representation of the first 30 hexadecimal digits of the SHA256 hash of the UTF-8 encoded bytes of `strippedID`.

Exceptions:

> If the following conditions occur in the process, prid generation fails:

1. Leading 6 characters of PersonIdentifier does not match regexp `^[A-Za-z]{2}[\/](SE|se)[\/]$`
2. `normalizedID` < 8 characters (not counting “-“ (hyphen) characters).

Collision resistance:

All eIDAS PersonIdentifier attributes are required to be unique. That uniqueness is preserved in this algorithm based on the assumption that distinguishing symbols in the PersonIdentifier are represented exclusively by case insensitive letters a-z and numbers 0-9. If this is not a case for the PersonIdentifier provided by a particular country, then this algorithm should not be selected due to the risk of ID collisions.

In case the normalized identifier string exceeds 30 characters this algorithm falls back on providing a 30 hex digit representation of a concatenated SHA-2 hash value. The collision probability for such identifier among 2 users is 1 in 1,25 * 10\^36 (Calculated as 16\^30 – 16\^29 since first digit can not be 0). For a population of 100 million people the probability of a collision is approximately 1 in 2.5 * 10\^20 or 1 in 250 trillion countries of that population size[1].

Countries typically do not use ID attributes that exceed 30 characters in size, but it cannot be ruled out that some countries will generate an ID for cross-border use that is different from a national ID and that that ID may exceed 30 characters. This algorithm assumes that the collision resistance provided is sufficient given both the low probability combined with the fact that only a very small fraction of users from any country outside of Sweden is likely to authenticate to a Swedish e-service. A collision among all citizens would in most cases not be security critical since the unique eIDAS ID in its original form also is present in the assertion to the service provider, which guarantees that a transaction is traceable back to the right individual.

In case these properties are not enough to guarantee sufficient collision resistance, the algorithm colresist-eIDAS should be used.

Examples:

PersonIdentifier	**Resulting prid**
NO/SE/05068907693	NO:05068907693
DK/SE/09208-2002-2-194967071622	DK:09208-2002-2-194967071622
UK/DK/1234567890	UK:NULL (Failed: target country is not SE)
DE/SE/#12345-3456//ABC	DE:12345-3456-abc
DE/SE/aErf#(EAd9)	DE:0aerf-ead9

PersonIdentifier	Resulting prid
de/se/aErf#(EAd)	NULL (Failed: Less than 8 ID characters)
DE/SE/(1952 12 14-1122)	DE:19521214-1122
19521214-1122	NULL (Failed: Leading 6 character format error)
DE/SE/12345678901234567890123456789O1	DE:3b7184c0ceaf76a9607a31e4e1f87f

> [1]: Birthday paradox approximation p(*n*) ≈ *n*\^2 / 2*m*, where p(*n*) is the collision probability, *n* is the number of people and *m* is the number of possible ID combinations.

2.3.2. Algorithm: colresist-eIDAS

This algorithm is identical to default-eIDAS except that the hashed expression of the ID in case the normalized ID exceeds 30 characters, has higher collision resistance by expressing the ID in radix 36 instead of radix 16 (Hexadecimal)[1].

Name: `colresist-eIDAS`

Input values: Identical to `default-eIDAS`

Calculated values: Identical to `default-eIDAS`

Result:

If length of normalizedID < 10 characters:

> Identical to `default-eIDAS` result

If length of normalizedID 10 - 30 characters:

> Identical to `default-eIDAS` result

If length of normalizedID > 30 characters**:**

> Return the string representation of the first 30 radix 36 digits of the SHA256 hash of the UTF-8 encoded bytes of `strippedID`.

Exceptions: Identical to `default-eIDAS`

Collision resistance:

Expression of this ID in hashed form use 30 radix 36 symbols. This reduces the collision resistance among 2 users to 1 in 4.75 * 10\^46. The collision probability among 100 million users is reduced to approximately 1 in 10\^31.

Examples:

PersonIdentifier	Resulting prid
NO/SE/05068907693	NO:05068907693
DK/SE/09208-2002-2-194967071622	DK:09208-2002-2-194967071622
UK/DK/1234567890	UK:NULL (Failed: target country is not SE)

PersonIdentifier	Resulting prid
DE/SE/#12345-3456//ABC	DE:12345-3456-abc
DE/SE/aErf#(EAd9)	DE:0aerf-ead9
de/se/aErf#(EAd)	NULL (Failed: Less than 8 ID characters)
DE/SE/(1952 12 14-1122)	DE:19521214-1122
19521214-1122	NULL (Failed: Leading 6 character format error)
DE/SE/12345678901234567890123456789O1	DE:1hc3tpoleczqu3t8jz2995k2rq7nt8

> [1]: Radix 36 express values ranging from 0 to 36 through a single character using the symbols ”0123456789abcdefghijklmnopqrstuvwxyz” in the presented order.

2.3.3. Algorithm: special-characters-eIDAS

The default-eIDAS and the colresist-eIDAS algoritms are suitable when the base identifier from the authenticating country is constructed from digits and basic case-insensitive characters. These algoritms do not work on identifiers constructed as a Base64 string of binary data, such as a hash of another identifier.

The present algoritm is intended to be used where the base identifier contains case-sensitive characters and where characters other than a-z and 0-9 are used to add entropy to the identifier.

Name: `special-characters-eIDAS`

Input values: Identical to `default-eIDAS`.

Calculated values: Identical to `default-eIDAS` with the exception that `normalizedID` is not calculated and used.

Result:

> Return the string representation of the first 30 radix 36 digits of the SHA256 hash of the UTF-8 encoded bytes of `strippedID`.

Exceptions:

> If the following conditions occur in the process, prid generation fails:

1. Leading 6 characters of PersonIdentifier does not match regexp `^[A-Za-z]{2}[\/](SE|se)[\/]$`
2. `strippedID` < 16 characters.

Collision resistance: Identical to `colresist-eIDAS`.

Examples:

PersonIdentifier	Resulting prid
AT/SE/Zk2ME2pjxwzQOjVeFGeqSIage34=	AT:50bwytdle2mzexopcolmdhmhznihms

2.4. Algorithm Selection and Resulting pridPersistence Value

This section defines the current algorithm selection rules and the resulting pridPersistence value. These rules are processed in the presented order. The first rule where the present conditions matches all the matching rules is selected.

If the present conditions do not match any of the listed rules, then prid generation fails.

Rule 1	**Description**
Matching rule 1	Authenticated attributes are provided by an eIDAS node (proxy service).
Matching rule 2	Authenticated subject is a person and has a `PersonIdentifier` attribute.
Matching rule 3	Attributes provided by any of the countries listed in Appendix A.
Selected algorithm	`default-eIDAS`
pridPersistence value	A

Rule 2	**Description**
Matching rule 1	Authenticated attributes are provided by an eIDAS node (proxy service).
Matching rule 2	Authenticated subject is a person and has a `PersonIdentifier` attribute.
Matching rule 3	Attributes provided by any of the countries listed in Appendix B.
Selected algorithm	`default-eIDAS`
pridPersistence value	B

Rule 3	**Description**
Matching rule 1	Authenticated attributes are provided by an eIDAS node (proxy service).
Matching rule 2	Authenticated subject is a person and has a `PersonIdentifier` attribute.
Selected algorithm	`default-eIDAS`
pridPersistence value	C

3. References

[RFC2119]

> Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, March 1997.

[SAML2Core]

> OASIS Standard, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, March 2005.

[SAML-XSD]

> S. Cantor et al., SAML assertions schema. OASIS SSTC, March 2005. Document ID saml-schema-assertion-2.0. See http://www.oasisopen.org/committees/security/.

[XML-Schema]

> XML Schema Part 2: Datatypes Second Edition, W3C Recommendation, 28 October 2004. See http://www.w3.org/TR/xmlschema-2/.

[EidAttributes]

> Attribute Specification for the Swedish eID Framework.

[eIDAS-Attr]

> eIDAS SAML Attribute Profile, version 1.2, 21 May 2019.

4. Changes between versions

Changes between version 1.0 and version 1.1:

- Update of logotype and fixes of minor typos.
- Update of reference of eIDAS attribute profile to version 1.2.

Appendix A. Countries with pridPersistence of class A

The following countries provide an eIDAS PersonIdentifier attribute that has been determined to match pridPersistence level A:

ISO 3166	Name
DK	Denmark
NO	Norway
SE	Sweden

Appendix B. Countries with pridPersistence of class B

The following countries provide an eIDAS PersonIdentifier attribute that has been determined to match pridPersistence level B:

ISO 3166	Name
DE	Germany

Appendix C. PRID Algorithm implementations (Java)

default-eIDAS

```
private static final String personIdentifierPrefixRegexp = "^[A-Za-z]{2}[\\/](SE|se)[\\/]";

public String getPridIdentifierComponent(String personIdentifier) {
  if (personIdentifier == null) {
    return null;
  }
  if (!personIdentifier.substring(0, 6).matches(personIdentifierPrefixRegexp)) {
    return null;
  }
  // Get ID component without whitespace and non-printable characters
  String strippedID = personIdentifier.substring(6).replaceAll("\\s+", "");

  // Convert to lower case
  String normalizedID = strippedID.toLowerCase();

  // Replace sequences of non ID characters to "-"
  normalizedID = normalizedID.replaceAll("[^a-z0-9]+", "-");

  // Remove leading and trailing "-"
  normalizedID = normalizedID.replaceAll("^-+", "").replaceAll("-+$", "");
  if (normalizedID.replaceAll("-", "").length() < 8) {
    return null;
  }
  if (normalizedID.length() < 10) {
    normalizedID = "0000000000".substring(normalizedID.length()) + normalizedID;
  }
  if (normalizedID.length() > 30) {
    try {
      MessageDigest md = MessageDigest.getInstance("SHA-256");
      byte[] digest = md.digest(strippedID.getBytes(Charset.forName("UTF-8")));
      return new BigInteger(1, digest).toString(16).substring(0, 30);
    }
    catch (NoSuchAlgorithmException ex) {
      return null;
    }
  }
  return normalizedID;
}
```

colresist-eIDAS

```
private static final String personIdentifierPrefixRegexp = "^[A-Za-z]{2}[\\/](SE|se)[\\/]";

public static String getPridIdentifierComponent(String personIdentifier) {
  if (personIdentifier == null) {
    return null;
  }
  if (!personIdentifier.substring(0, 6).matches(personIdentifierPrefixRegexp)) {
    return null;
  }

  // Get ID component without whitespace and non-printable characters
```

```
  String strippedID = personIdentifier.substring(6).replaceAll("\\s+", "");

  // Convert to lower case
  String normalizedID = strippedID.toLowerCase();

  // Replace sequences of non ID characters to "-"
  normalizedID = normalizedID.replaceAll("[^a-z0-9]+", "-");

  // Remove leading and trailing "-"
  normalizedID = normalizedID.replaceAll("^-+", "").replaceAll("-+$", "");
  if (normalizedID.replaceAll("-", "").length() < 8) {
    return null;
  }
  if (normalizedID.length() < 10) {
    normalizedID = "0000000000".substring(normalizedID.length()) + normalizedID;
  }
  if (normalizedID.length() > 30) {
    try {
      MessageDigest md = MessageDigest.getInstance("SHA-256");
      byte[] digest = md.digest(strippedID.getBytes(Charset.forName("UTF-8")));
      return new BigInteger(1, digest).toString(36).substring(0, 30);
    }
    catch (NoSuchAlgorithmException ex) {
      return null;
    }
  }
  return normalizedID;
}
```

special-characters-eIDAS

```
private static final String personIdentifierPrifixRegexp = "^[A-Za-z]{2}[\\/](SE|se)[\\/]";

public String getPridIdentifierComponent(String personIdentifier) {
  if (personIdentifier == null) {
    return null;
  }
  if (!personIdentifier.substring(0, 6).matches(personIdentifierPrifixRegexp)) {
    return null;
  }

  // Get ID component without whitespace and non-printable characters
  String strippedID = personIdentifier.substring(6).replaceAll("\\s+", "");
  if (strippedID.length() < 16) {
    return null;
  }
  try {
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    byte[] digest = md.digest(strippedID.getBytes(Charset.forName("UTF-8")));
    return new BigInteger(1, digest).toString(36).substring(0, 30);
  }
  catch (NoSuchAlgorithmException ex) {
    return null;
  }
}
```