VOTING Stimmregister

Digital Signature

Author	Abraxas Informatik AG
Classification	public
Version	1.1
Date	31. Jan 2024

Für die digitale Schweiz. Mit Sicherheit.

abraxas

Contents

1. Key Management

The signature payload must be encrypted using a secure signature method and meet the requirements of the cryptography policy. The following table defines the specifications for the HSM signature key:

	Algorithm	Basis for decision-making
Signature Algorithm	ECDSA	The ECDSA algorithm meets all security requirements for the application and cryptography policies. ECDSA is significantly faster at signing (.NET Support)
Elliptic Curve	NIST P-384	NIST-specified curve (.NET Support)
Hash Algorithm	SHA2-384	Fast and secure hashing algorithm (.NET Support)

1.1 Key Rotation

Key rotation is applied when a signing key is retired and replaced by generating a new cryptographic key on the HSM. Keys are rotated on a regular basis once per year but can be changed more frequently if needed. When signature keys are rotated, the following tasks must be scheduled:

- **HSM** The rotated private key on the HSM is deactivated, while the public key remains accessible for verification of older signatures for backward compatibility.
- **HSM** The public key is backed-up (exported) to ensure signature verification in case of HSM service failure
- **HSM** A new ECDSA key pair is created on the HSM with the CK_LABEL and the new version added as postfix:
    - Syntax: {CK_LABEL}_V{VERSION}
    - Sample Version 2: VOSR_ECDSA_PUBLIC_KEY_PRE_V2 / VOSR_ECDSA_PRIVATE_KEY_PRE_V2
- **SERVICE** A new signature version is created on the application side for BFS, person and version signatures.
- **SERVICE** The new version is configured to reference the new CK_LABEL identifications.

1.2 Backup Public Key

The signing public keys are backed up to be able to verify signatures that have been created at any point in time. This includes public keys being rotated meanwhile as well as active keys pairs. To ensure integrity of the backed up public keys, the key parameters are stored in a vault.

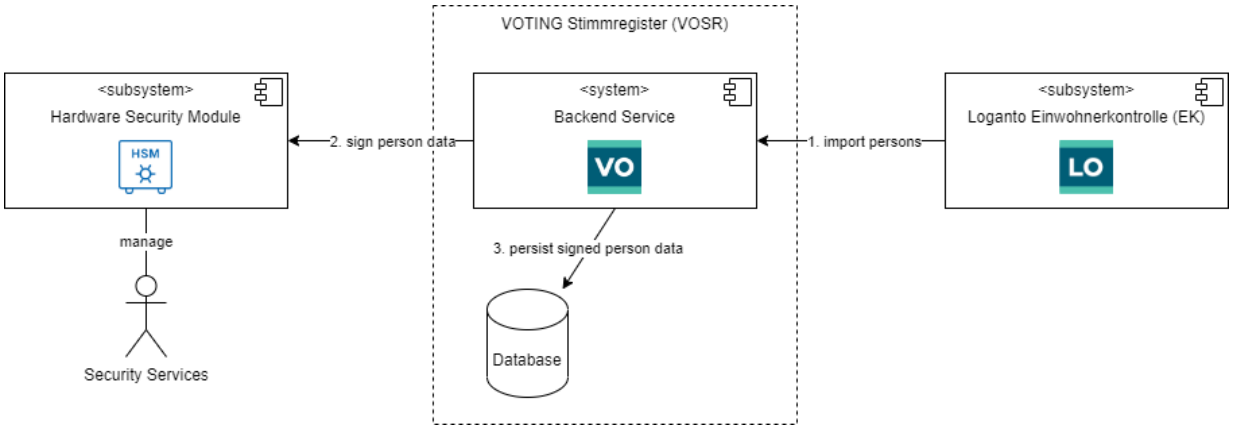
1.2.1 Key Overview

Identifier (CKA Label)	Key	Algorithm	Curve	Version
VOSR_ECDSA_PUBLIC_KEY_PRE	Public Key	ECDSA	NIST-P384	V1
VOSR_ECDSA_PUBLIC_KEY_PRO	Public Key	ECDSA	NIST-P384	V1

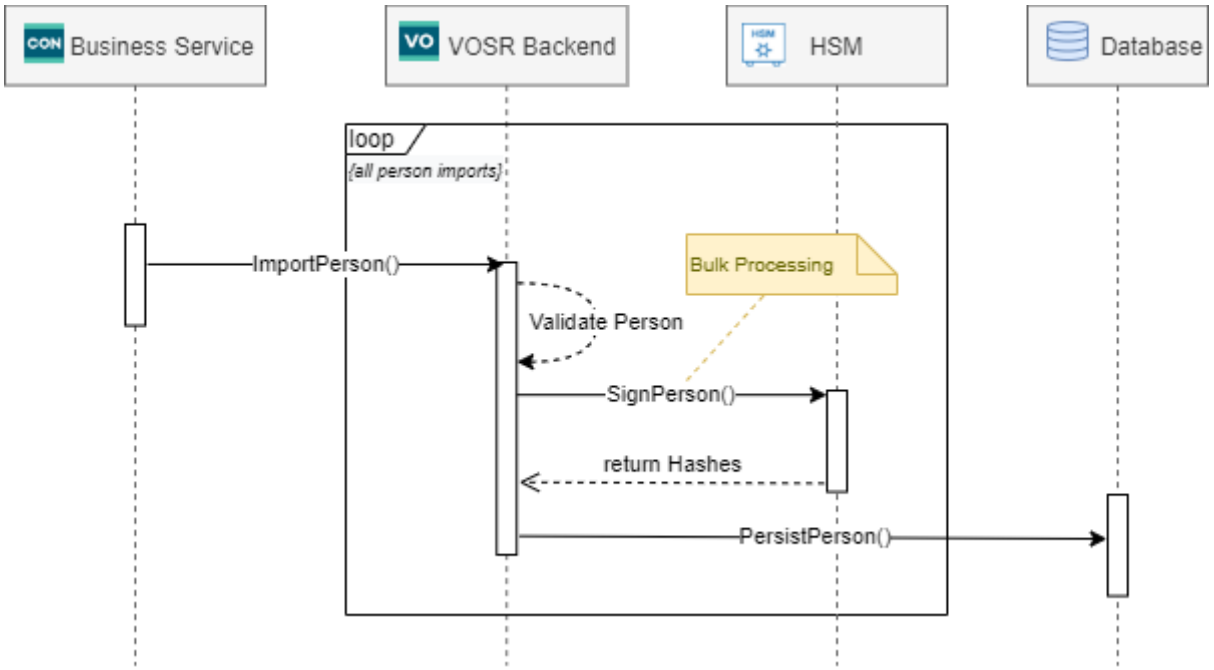
2. Person Signature

2.1 System Overview

2.1.1 Block Diagram



2.1.2 Sequence Diagram



2.2 Signature Specification (V1)

2.2.1 Signature Generation

The signature is built based on a concatenation of byte arrays each representing data to prove the person's integrity. The signature generation and attribute concatenation follows a predefined ordering as defined by the position indicator.

```
// By attribute name: Create byte array from defined attributes to be signed
byte[] bytesToSign = CONCAT(SignatureVersion, KeyId, Payload)

// By attribute position: Create byte array from defined attributes to be signed
byte[] bytesToSign = CONCAT(1, 2, 3)

// Sign byte array by trusted service
byte[] signature = SIGN(bytesToSign)
```

2.2.2 Signature Normalization

All attributes of different data types must satisfy the normalization idempotency to ensure that the signatures can be generated and verified at any point in time. The following rule sets must be applied for different datatypes:

Type [in]	**Normalization Rule** ***<Pseudocode Example>***	**Type [out]**	**Sample [in]**	**Sample [out]**
byte array	value	byte[]	0xAABF23	{ 0xAA, 0xBF, 0x23 }
byte	new byte[] { value }	byte[]	0xFA	{ 0xFA }
boolean	true → new byte[] { 0x01 } false → new byte[] { 0x00 }	byte[]	true	{ 0x01 }
int32	value.Int32BigEndian()	byte[]	277	{ 0x01, 0x15 }
int64	value.Int64BigEndian()	byte[]	12345678901234567890	{ 0xAB, 0x54, 0xA9, 0x8C, 0xEB, 0x1F, 0x0A, 0xD2 }
string	UTF8.GetBytes(value)	byte[]	"Hi"	{ 0x48, 0x69 }
guid	According to RFC4122 value.TryWriteBytes(span, true, out var written)	byte[]	5c41a76f-a7c0-43c8-8d1b-1bba40bcd501	{ 0x5c, 0x41, 0x7a, …, 0xd5, 0x01 }
dateTime	DateTimeOffset(value) .ToUnixTimeMilliseconds().Int64BigEndian()	byte[]	15.11.1980 → 343094400000	{ 0x00, 0x00, 0x00, 0x4F, 0xE2, 0x05, 0x14, 0x00 }
<null>	treat as empty byte array	byte[]	null	{ }
<empty string>	treat as empty byte array	byte[]	""	{ }

2.2.3 Signature Attributes

The following table provides an overview of all attributes to generate the signature:

Attribute	Position	Data type	Data type (original)	Value range (original)	Description	Normalization
SignatureVersion	1	Byte (fix 0x01)	byte	1...255	The signature version represents the specification that must be applied during creation and verification of signatures.	n/a
SignatureKeyId	2	byte array	string	1...n	The key id corresponds to the unique identifier CKA_LABEL for a specific slot of the HSM. The key id may change when private key rotating is applied.	UTF8
Business Payload *<1..n attributes>*	3	byte array	byte array	Application-dependent	The business payload contains all business-relevant person data as described in chapter "Signature Business Payload"	n/a

2.2.4 Signature Business Payload

The payload is represented as a byte array and based on a concatenation of business relevant person attributes. The payload is part of the person's signature as specified in chapter "Signature Attributes". The signature generation and attribute concatenation follows a predefined ordering as defined by the position indicator.

```
// Create byte array from defined person data attributes by attribute name
byte[] payload = CONCAT(Id, RegisterId, Vn, ..., PeopleCircleId,
PeopleCircleName)

// Create byte array from defined person data attributes by attribute position
byte[] payload = CONCAT(1, 2, 3, ..., 56)
```

Codeblock 1 Signature Generation (Person Data Payload)

Detailed description for each person attribute can be found from the Person Data Fields documentation. The following table provides an overview of all relevant attributes that must be part of the final signature to prove the person's integrity.

Attribute	Position	Data type	Data type (original)	Value range (original)	Remark	Normalization
Id	1	Byte array	guid	128 bit	n/a	RFC4122
RegisterId	2	Byte array	guid	128 bit	n/a	RFC4122
Vn	3	Byte array	int32	43 bit	n/a	Big-Endian
DomainOfInfluenceId	4	Byte array	int32	32 bits	n/a	Big-Endian

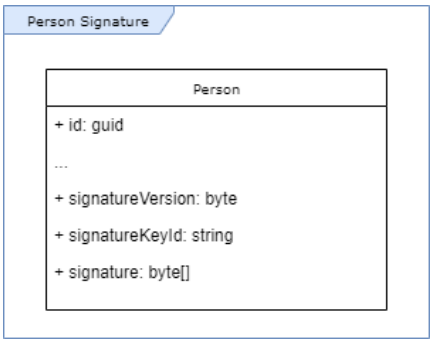
Attribute	Position	Data type	Data type (original)	Value range (original)	Remark	Normalization
OfficialName	5	Byte array	string	1..n	n/a	UTF8
FirstName	6	Byte array	string	1..n	n/a	UTF8
DateOfBirth	7	Byte array	dateTime	64 bit	n/a	Big-Endian
DateOfBirthAdjusted	8	Byte	boolean	0x00 / 0x01	n/a	n/a
Religion	9	Byte array	int32	0..122	n/a	Big-Endian
IsSwissAbroad	10	Byte	boolean	0x00 / 0x01	n/a	n/a
RestrictedVotingAnd ElectionRightFederation	11	Byte array	boolean	0x00 / 0x01	n/a	n/a
Country	12	Byte array	string	16 bit	n/a	UTF8
SwissAbroadEvotingFlag	13	Byte array	boolean	0x00 / 0x01	n/a	n/a
MunicipalityName	14	Byte array	string	1..n	n/a	UTF8
MunicipalityId	15	Byte array	int32	1..9999	n/a	Big-Endian
TypeOfResidence	16	Byte array	int32	1..3	n/a	Big-Endian
ResidencePermit	17	Byte array	string	1..n	n/a	UTF8
ResidencePermitValidFrom	18	Byte array	dateTime	64 bit	n/a	Big-Endian
ResidencePermitValidTill	19	Byte array	dateTime	64 bit	n/a	Big-Endian
ContactAddressExtensionLine1	20	Byte array	string	1..n	n/a	UTF8
ContactAddressExtensionLine2	21	Byte array	string	1..n	n/a	UTF8
ContactAddressStreet	22	Byte array	string	1..n	n/a	UTF8
ContactAddressHouseNumber	23	Byte array	string	1..n	n/a	UTF8
ContactAdressDwellingNumber	24	Byte array	string	1..n	n/a	UTF8
ContactAddressHouse NumberAddition	25	Byte array	string	1..n	n/a	UTF8

Attribute	Position	Data type	Data type (original)	Value range (original)	Remark	Normalization
ContactAddressPostOffice BoxText	26	Byte array	string	1..n	n/a	UTF8
ContactAddressPostOffice BoxNumber	27	Byte array	int32	32 bits	n/a	Big-Endian
ContactAddressLine1	28	Byte array	string	1..n	n/a	UTF8
ContactAddressLine2	29	Byte array	string	1..n	n/a	UTF8
ContactAddressLine3	30	Byte array	string	1..n	n/a	UTF8
ContactAddressLine4	31	Byte array	string	1..n	n/a	UTF8
ContactAddressLine5	32	Byte array	string	1..n	n/a	UTF8
ContactAddressLine6	33	Byte array	string	1..n	n/a	UTF8
ContactAddressLine7	34	Byte array	string	1..n	n/a	UTF8
ContactAddressTown	35	Byte array	string	1..n	n/a	UTF8
ContactAddressLocality	36	Byte array	string	1..n	n/a	UTF8
ContactAddressZipCode	37	Byte array	string	1..n	n/a	UTF8
ResidenceAddressStreet	38	Byte array	string	1..n	n/a	UTF8
ResidenceAddressHouseNumber	39	Byte array	string	1..n	n/a	UTF8
ResidenceAddressDwellingNumber	40	Byte array	string	1..n	n/a	UTF8
ResidenceAddressTown	41	Byte array	string	1..n	n/a	UTF8
ResidenceCountry	42	Byte array	string	1..n	n/a	UTF8
ResidenceAddressZipCode	43	Byte array	string	1..n	n/a	UTF8
ResidenceCantonAbbreviation	44	Byte array	string	1..n	n/a	UTF8
MoveInArrivalDate	45	Byte array	dateTime	64 bit	n/a	Big-Endian
SendVotingCardsToDomain OfInfluenceReturnAddress	46	Byte	boolean	0x00 / 0x01	n/a	n/a

Attribute	Position	Data type	Data type (original)	Value range (original)	Remark	Normalization
PoliticalCircleId	47	Byte array	string	1..n	n/a	UTF8
PoliticalCircleName	48	Byte array	string	1..n	n/a	UTF8
CatholicCircleId	49	Byte array	string	1..n	n/a	UTF8
CatholicCircleName	50	Byte array	string	1..n	n/a	UTF8
EvangelicCircleId	51	Byte array	string	1..n	n/a	UTF8
EvangelicCircleName	52	Byte array	string	1..n	n/a	UTF8
SchoolCircleId	53	Byte array	string	1..n	n/a	UTF8
SchoolCircleName	54	Byte array	string	1..n	n/a	UTF8
TrafficCircleId	55	Byte array	string	1..n	n/a	UTF8
TrafficCircleName	56	Byte array	string	1..n	n/a	UTF8
ResidentialDistrictCircleId	57	Byte array	string	1..n	n/a	UTF8
ResidentialDistrictCircleName	58	Byte array	string	1..n	n/a	UTF8
PeopleCircleId	59	Byte array	string	1..n	n/a	UTF8
PeopleCircleName	60	Byte array	string	1..n	n/a	UTF8
DeleteAt	61	Byte array	dateTime	64 bit	n/a	Big-Endian
CantonBfs	62	Byte array	int16	1..26	n/a	Big-Endian

2.2.5 Signature Persistence

The signature related data attributes are stored as part of the person data model. For simplicity the common person attributes are excluded from the following data model:



2.2.6 Signature Validation

Before creating or saving a new version to a filter, the signature of each person must be verified. Only the signatures of persons who are assigned to the version via the filter criteria must be verified. If at least 1 signature fails, a security log must be written and an error message returned to the user.

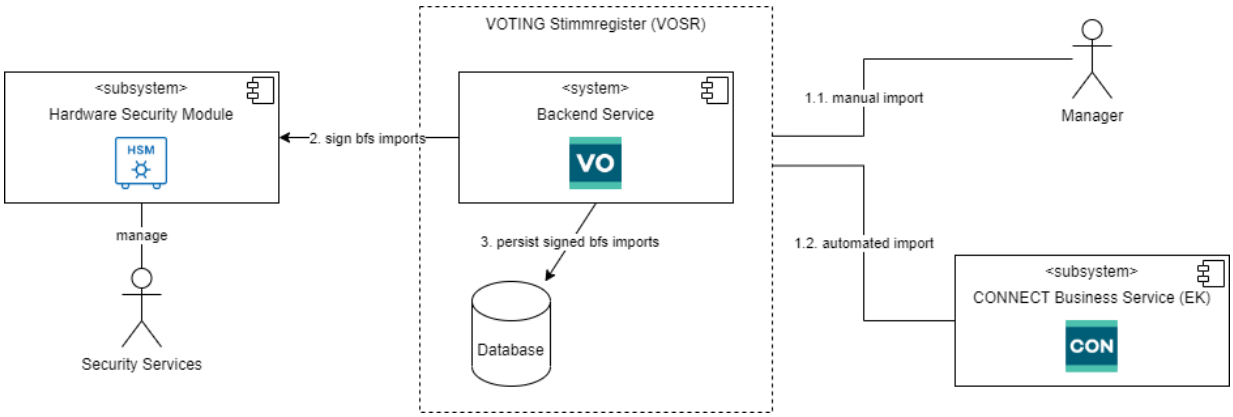
3. BFS Integrity Signature

While importing, in order to determine the completeness of all imported persons or domain of influence, an integrity record will be created to verify that a all persons or domain of influence of a BFS are correct.

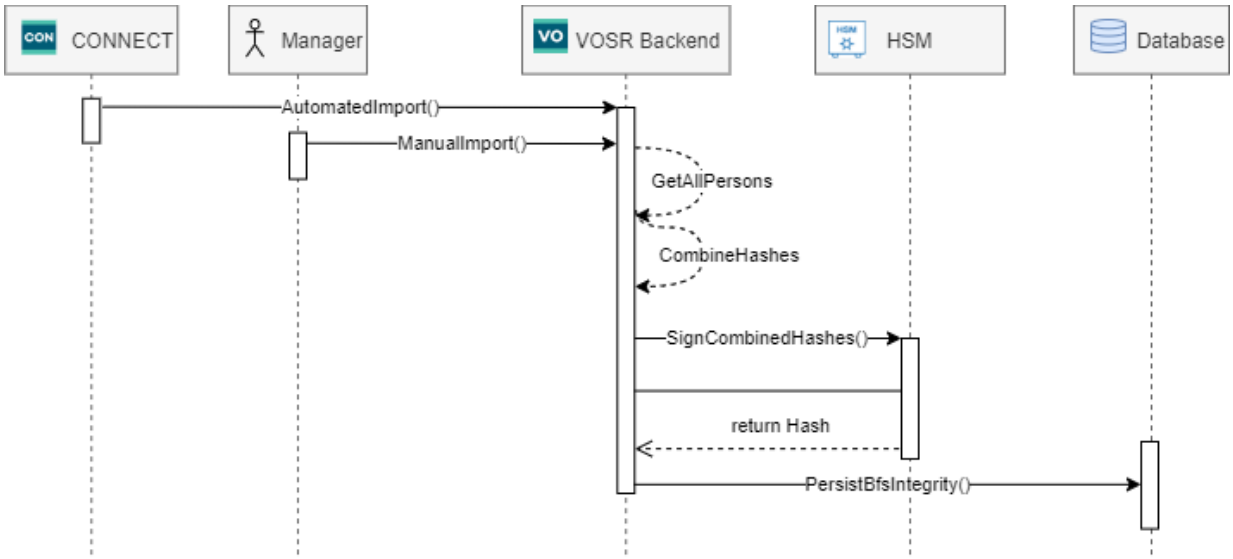
The integrity signature process is identical for persons and domain of influence imports. Since they're handled the same way only persons will be reflected in further descriptions for simplicity.

3.1 System Overview

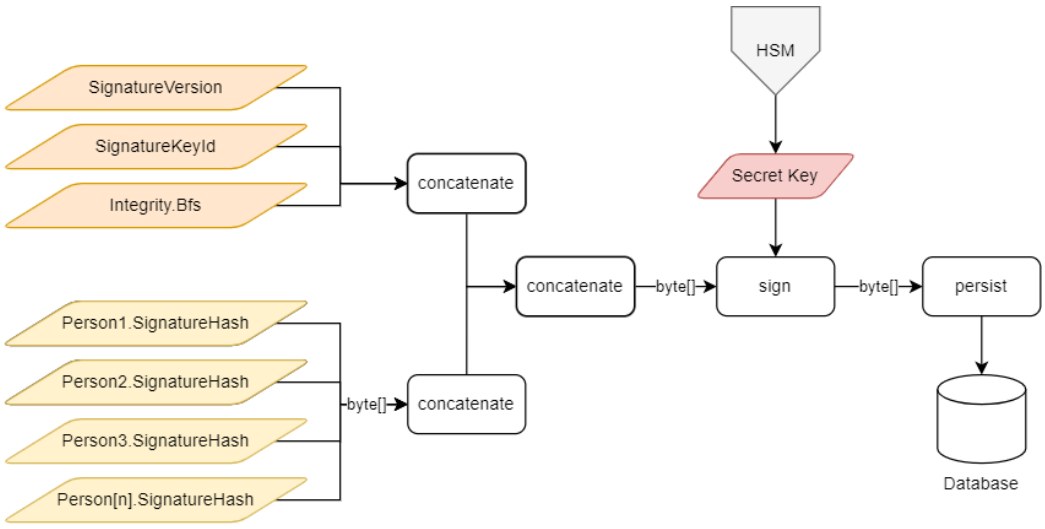
3.1.1 Block Diagram



3.1.2 Sequence Diagram



3.2 Signature Specification (V1)



3.2.1 Signature Generation

The signature for the BFS data is built based on the concatenation of the existing person signatures. By re-using the already proved integrity of each person, the BFS integrity can be defined on top. The BFS integrity is always based on the latest revision of persons being imported for a specific BFS range. The signature generation and attribute concatenation follow a predefined ordering as defined by the position indicator.

```
// By attribute name: Create byte array from defined attributes to be signed
byte[] bytesToSign = CONCAT(SignatureVersion, SignatureKeyId, Bfs, LastUpdated,
Payload)

// By attribute position: Create byte array from defined attributes to be signed
byte[] bytesToSign = CONCAT(1, 2, 3, 4, 5)

// Sign byte array by trusted service
byte[] signature = SIGN(bytesToSign)
```

3.2.2 Signature Normalization

All attributes of different data types must satisfy the normalization idempotency to ensure that the signatures can be generated and verified at any point in time. The following rule sets must be applied for different datatypes:
Refer to:

- Person Signature
- DOI Signature

3.2.3 Signature Attributes

The following table provides an overview of all attributes to generate the signature:

Attribute	Position	Data type	Data type (original)	Value range (original)	Description	Normalization
SignatureVersion	1	Byte (fix 0x01)	byte	1...255	The signature version represents the specification that must be applied during creation and verification of signatures.	n/a
SignatureKeyId	2	byte array	string	1...n	The key id corresponds to the unique identifier CKA_LABEL for a specific slot of the HSM. The key id may change when private key rotating is applied.	UTF8
Bfs	3	byte array	string	1...9999	The BFS (Bundesamt für Statistik) number that identifies the data import for a specific scope.	UTF8
LastUpdated	4	byte array	dateTime	64 bit	Defines the time when the last successful import was performed and an integrity signature is created.	Big-Endian
BFS Signature Payload	5	byte array	byte array	Application-dependent	The business payload contains all business-relevant BFS data as described in chapter "BFS Signature Payload"	n/a

3.2.4 BFS Signature Payload

The person and DOI signature payload is built equally but differs in the target entities and attributes to be included. The following two chapters describe how to build the payload for either persons and DOIs.

3.2.4.1 Person Signature Payload

The person's payload is represented as a byte array and based on a concatenation of signatures from all person's latest revisions. The payload is part of the BFS signature as specified in chapter "Signature Attributes". The signature generation and attribute concatenation follows a predefined ordering as defined by the position indicator.

```
// Create byte array from defined person data attributes by attribute name
byte[] payload = CONCAT(SignatureHashPerson1, SignatureHashPerson2, ...,
SignatureHashPerson_n)
```

Detailed description for attributes can be found on the corresponding documentation: Person
The following table provides an overview of all relevant attributes that must be part of the final signature to prove the entities integrity.

Attribute	Position	Data type	Data type (original)	Value range (original)	Remark	Normalization
Person.SignatureHash	1...n	Byte array	byte array	Depends on the number of assigned persons for the target BFS scope.	n/a	n/a

3.2.4.2 DOI Signature Payload

The DOI's payload is represented as a byte array and based on a concatenation of signatures from all DOI's latest revisions. The payload is part of the BFS signature as specified in chapter "Signature Attributes". The signature generation and attribute concatenation follows a predefined ordering as defined by the position indicator.

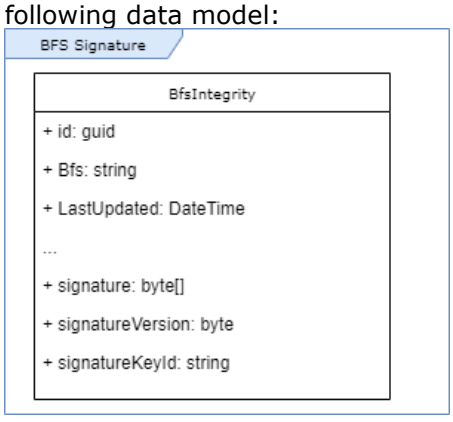
```
// Create byte array from defined domain of influence data attributes by
attribute name
byte[] payload = CONCAT_DELIMITED(SignatureHashDoi1, SignatureHashDoi2, ...,
SignatureHashDoi_n)
```

Detailed description for attributes can be found in the corresponding documentations: Domain of Influence - SSR
The following table provides an overview of all relevant attributes that must be part of the final signature to prove the entities integrity.

Attribute	Position	Data type	Data type (original)	Value range (original)	Remark	Normalization
DomainOfInfluence.SignatureHash	1...n	Byte array	byte array	Depends on the number of assigned DOIs for the target BFS scope.	n/a	n/a

3.2.5 Signature Persistence

The signature related data attributes for person and DOI are both stored as part of the integrity data model. For simplicity the common integrity attributes are excluded from the following data model:



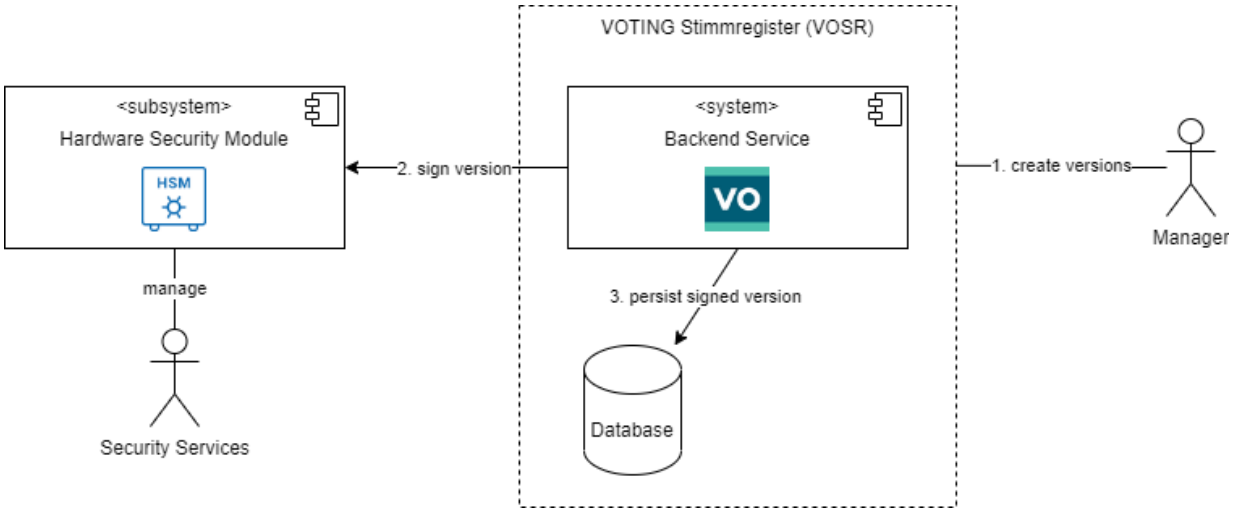
3.3 Signature Validation

The integrity of Person and DOI import signatures must be checked in a 24h cycle. Errors in the signatures are reported via the standard monitoring and alerting.

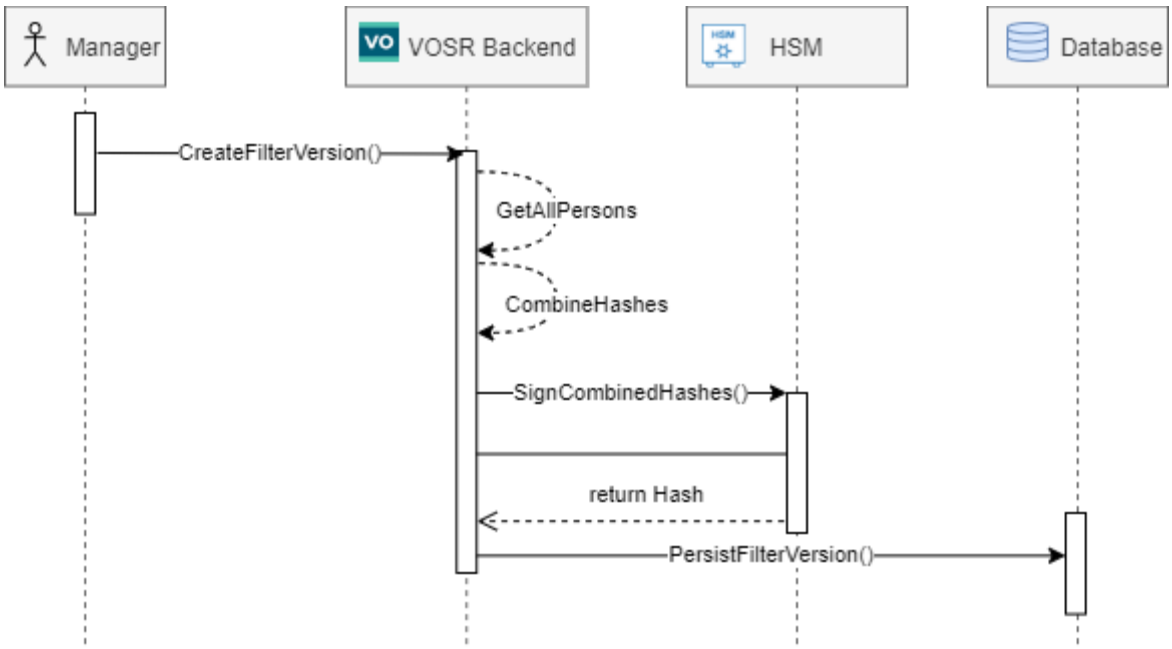
4. Version Signature

4.1 System Overview

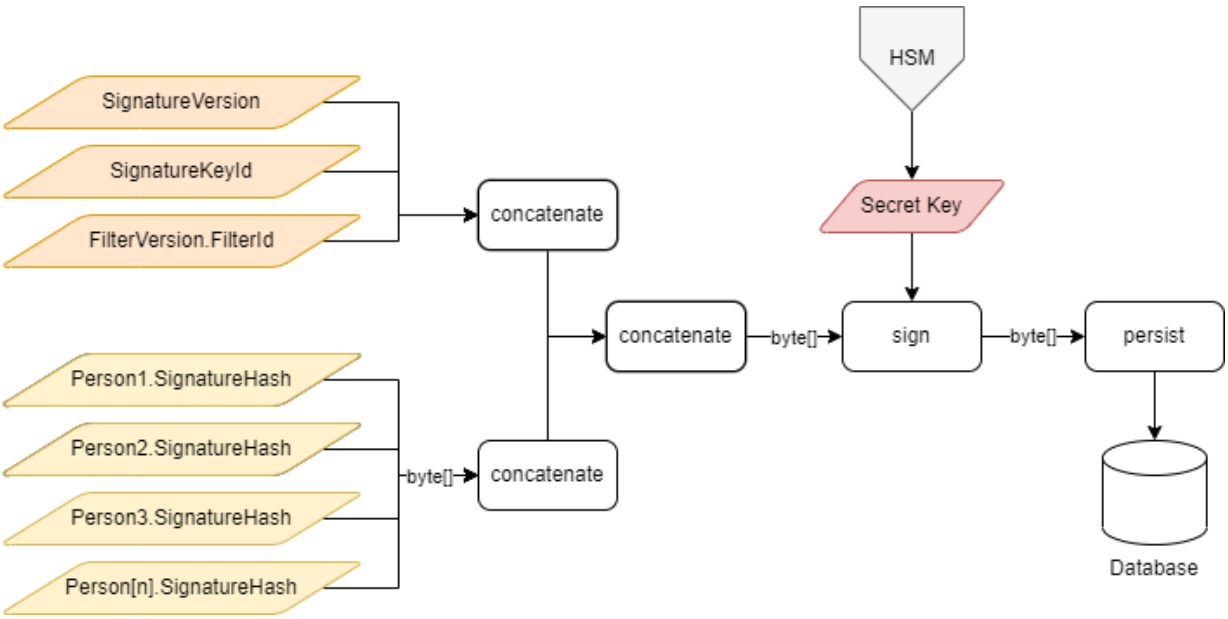
4.1.1 Block Diagram



4.1.2 Sequence Diagram



4.2 Signature Specification (V1)



4.2.1 Signature Generation

The signature for a filter version is built based on the concatenation of the existing person signatures. By re-using the already proved integrity of each person, the filter version integrity can be defined on top. The signature generation and attribute concatenation follow a predefined ordering as defined by the position indicator.

```
// By attribute name: Create byte array from defined attributes to be signed
byte[] bytesToSign = CONCAT(SignatureVersion, SignatureKeyId, FilterId, Payload)

// By attribute position: Create byte array from defined attributes to be signed
byte[] bytesToSign = CONCAT(1, 2, 3, 4, 5)

// Sign byte array by trusted service
byte[] signature = SIGN(bytesToSign)
```

4.2.2 Signature Normalization

All attributes of different data types must satisfy the normalization idempotency to ensure that the signatures can be generated and verified at any point in time. The rule sets as defined in chapter Person Signature must be applied for all datatypes.

4.2.3 Signature Attributes

The following table provides an overview of all attributes to generate the signature:

Attribute	Position	Data type	Data type (original)	Value range (original)	Description	Normalization
SignatureVersion	1	Byte (fix 0x01)	byte	1...255	The signature version represents the specification that must be applied during creation and verification of signatures.	n/a
SignatureKeyId	2	byte array	string	1...n	The key id corresponds to the unique identifier CKA_LABEL for a specific slot of the HSM. The key id may change when private key rotating is applied.	UTF8
FilterId	3	byte array	guid	128 bit	The unique id of the filter to which the version belongs to.	UTF8
Person Signature Payload	4	byte array	byte array	Application-dependent	The business payload contains all business-relevant person data as described in chapter "Person Signature Payload"	n/a

4.2.4 Person Signature Payload

The payload is represented as a byte array and based on a concatenation of signatures from all assigned persons. The persons should be ordered by bfs then by id, to get a reproducible payload.
The payload is part of the version's signature as specified in chapter "Signature Attributes".
The signature generation and attribute concatenation follows a predefined ordering as defined by the position indicator.

```
// Create byte array from defined person data attributes by attribute name
byte[] payload = CONCAT_DELIMITED(SignatureHashPerson1, SignatureHashPerson2,
..., SignatureHashPerson_n)
```

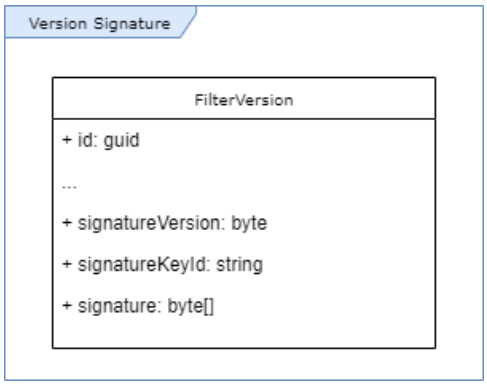
Codeblock 2 Signature Generation (Person Data Payload)

Detailed description for each person attribute can be found in the Person Data Fields documentation. The following table provides an overview of all relevant attributes that must be part of the final signature to prove the person's integrity.

Attribute	Position	Data type	Data type (original)	Value range (original)	Remark	Normalization
Person.SignatureHash	1...n	Byte array	byte array	Depending on the number of people to the version	n/a	Ordered by BFS then by Id

4.2.5 Signature Persistence

The signature related data attributes are stored as part of the filter version data model. For simplicity the common filter version attributes are excluded from the following data model:



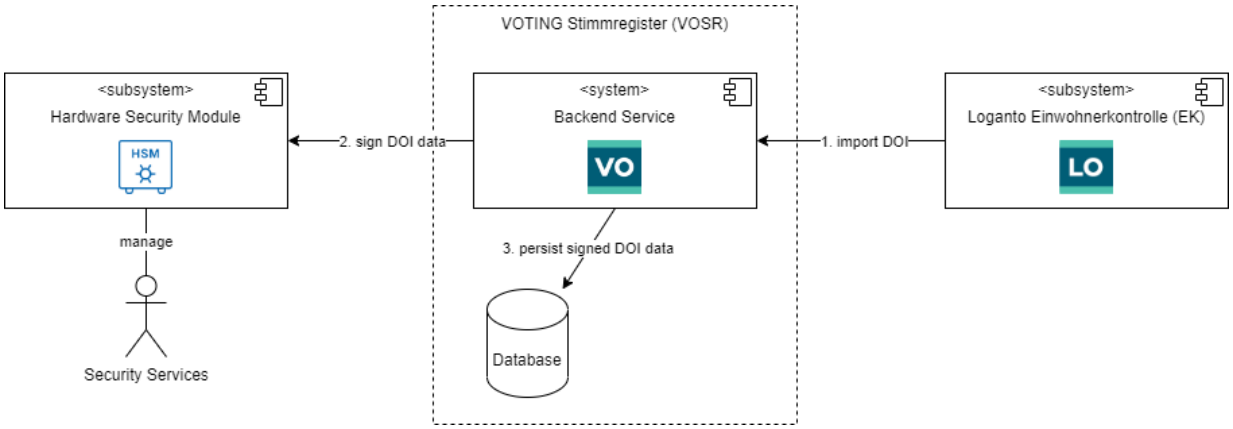
4.2.6 Signature Validation

Before each CSV or eCH-0045 export, the signature of the exported version must be verified. If the version signature fails, the export must be aborted, a security log must be written, and an error message must be displayed to the user. The same verification must also be applied for automated exports that are performed via API interfaces.

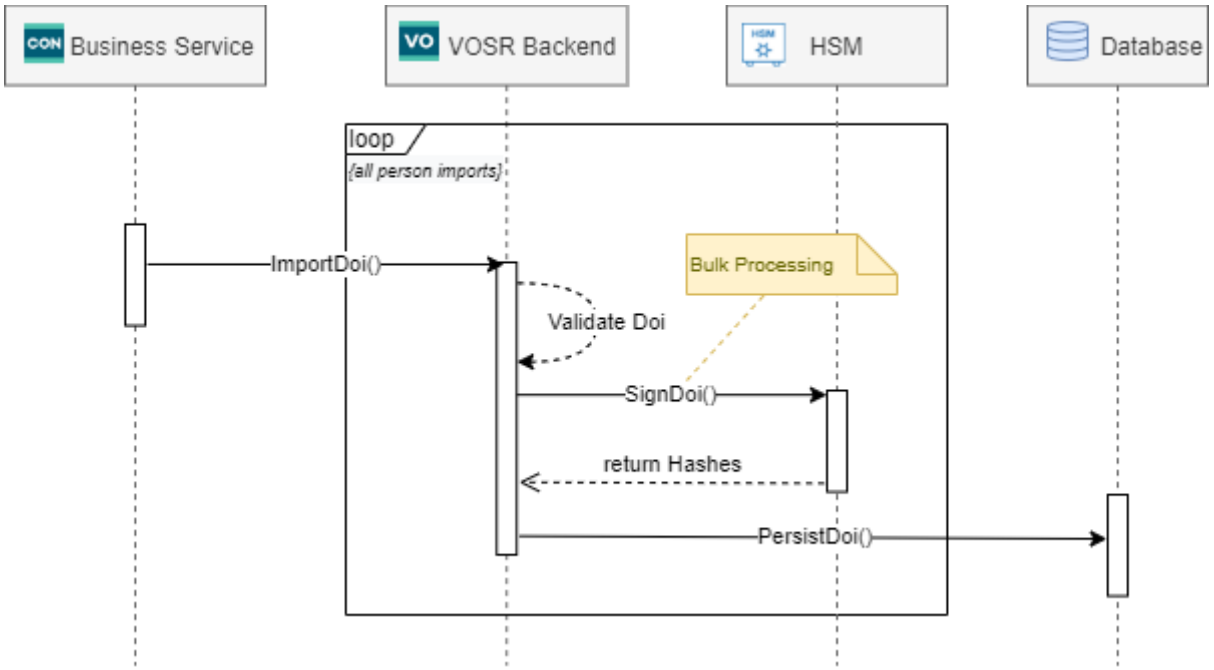
5. DOI Signature

5.1 System Overview

5.1.1 Block Diagram



5.1.2 Sequence Diagram



5.2 Signature Specification (V1)

5.2.1 Signature Generation

The signature is built based on a concatenation of byte arrays each representing data to prove the domain of influence's (DOI) integrity. The signature generation and attribute concatenation follow a predefined ordering as defined by the position indicator.

```
// By attribute name: Create byte array from defined attributes to be signed
byte[] bytesToSign = CONCAT(SignatureVersion, KeyId, Payload)

// By attribute position: Create byte array from defined attributes to be signed
byte[] bytesToSign = CONCAT(1, 2, 3)

// Sign byte array by trusted service
byte[] signature = SIGN(bytesToSign)
```

5.2.2 Signature Normalization

All attributes of different data types must satisfy the normalization idempotency to ensure that the signatures can be generated and verified at any point in time. The rule sets as defined in chapter Person Signature must be applied for all datatypes.

5.2.3 Signature Attributes

The following table provides an overview of all attributes to generate the signature:

Attribute	Position	Data type	Data type (original)	Value range (original)	Description	Normalization
SignatureVersion	1	Byte (fix 0x01)	byte	1...255	The signature version represents the specification that must be applied during creation and verification of signatures.	n/a
SignatureKeyId	2	byte array	string	1...n	The key id corresponds to the unique identifier CKA_LABEL for a specific slot of the HSM. The key id may change when the private key is being rotated.	UTF8
Business Payload *<1..n attributes>*	3	byte array	byte array	Application-dependent	The business payload contains all business-relevant DOI data as described in chapter "Signature Business Payload"	n/a

5.2.4 Signature Business Payload

The payload is represented as a byte array and based on a concatenation of business relevant DOI attributes. The payload is part of the DOI's signature as specified in chapter "Signature Attributes". The signature generation and attribute concatenation follows a predefined ordering as defined by the position indicator.

```
// Create byte array from defined DOI data attributes by attribute name
byte[] payload = CONCAT(Id, DomainOfInfluence, MunicipalityId, ...,
PeopleCouncilCircleId, PeopleCouncilCircleName)

// Create byte array from defined DOI data attributes by attribute position
byte[] payload = CONCAT(1, 2, 3, ..., 21)
```

Codeblock 3 Signature Generation (Person Data Payload)

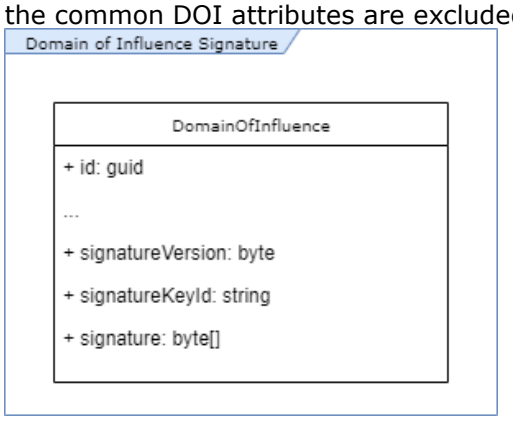
Detailed description for each DOI attribute can be found in the DOI Data Fields documentation. The following table provides an overview of all relevant attributes that must be part of the final signature to prove the DOI's integrity.

Attribute	Position	Data type	Data type (original)	Value range (original)	Remark	Normalization
Id	1	Byte array	guid	128 bit	n/a	RFC4122
DomainOfInfluenceId	2	Byte array	string	1..n	n/a	UTF8
MunicipalityId	3	Byte array	int32	32 bit	n/a	Big-Endian
Street	4	Byte array	string	1..n	n/a	UTF8
HouseNumber	5	Byte array	string	1..n	n/a	UTF8
HouseNumberAddition	6	Byte array	string	1..n	n/a	UTF8
SwissZipCode	7	Byte array	int32	32 bit	n/a	Big-Endian
IsPartOfPoliticalMunicipality	8	Byte array	boolean	8 bit	n/a	Unsigned integer
Town	9	Byte array	string	1..n	n/a	UTF8
PoliticalCircleId	10	Byte array	string	1..n	n/a	UTF8
PoliticalCircleName	11	Byte array	string	1..n	n/a	UTF8
CatholicChurchCircleId	12	Byte array	string	1..n	n/a	UTF8
CatholicChurchCircleName	13	Byte array	string	1..n	n/a	UTF8
EvangelicChurchCircleId	14	Byte array	string	1..n	n/a	UTF8
EvangelicChurchCircleName	15	Byte array	string	1..n	n/a	UTF8

Attribute	Position	Data type	Data type (original)	Value range (original)	Remark	Normalization
SchoolCircleId	16	Byte array	string	1..n	n/a	UTF8
SchoolCircleName	17	Byte array	string	1..n	n/a	UTF8
TrafficCircleId	18	Byte array	string	1..n	n/a	UTF8
TrafficCircleName	19	Byte array	string	1..n	n/a	UTF8
ResidentialDistrictCircleId	20	Byte array	string	1..n	n/a	UTF8
ResidentialDistrictCircleName	21	Byte array	string	1..n	n/a	UTF8
PeopleCouncilCircleId	22	Byte array	string	1..n	n/a	UTF8
PeopleCouncilCircleName	23	Byte array	string	1..n	n/a	UTF8

5.2.5 Signature Persistence

The signature related data attributes are stored as part of the DOI data model. For simplicity the common DOI attributes are excluded from the following data model:



5.2.6 Signature Validation

The integrity of the domain of influence signatures must be checked in a 24h cycle. Errors in the signatures are reported via the standard monitoring and alerting.

6. Signature Verification

A set of process flows in VOTING Stimmregister require that the integrity of the underlying person register is guaranteed at the time of executing specific processes. This requires signature verification at various levels. The processes and necessary signature checks are described below.

6.1 Verification Types

The integrity of the system can be checked at various times using the existing signatures, which are available at different levels. The signature checks are shown below. The in- and out of scope provide further information about which areas are covered by the corresponding check.

Identification	Verification Type	In scope	Out of scope	Specification
PER_SIG_VERIFY	Person Signature	• Detect manipulations on single person objects • Detect newly added person objects through unauthorized parties	Deleted person objects	Person Signature
DOI_SIG_VERIFY	DOI Signature	1. Detect manipulations on single DOI objects 2. Detect newly added DOI objects through unauthorized parties	Deleted DOI objects	DOI Signature
VER_SIG_VERIFY	Version Signature	• Detect manipulations on filter version person collection • Detect added person objects through unauthorized parties • Detect deleted person objects through unauthorized parties • Detect manipulated signatures on single person objects	Manipulations on single person objects	Version Signature
PER_BFS_VERIFY	Person BFS Integrity	• Detect manipulations on live data view (see following points) • Detect added person objects through unauthorized parties • Detect deleted person objects through unauthorized parties • Detect manipulated signatures on single person objects	Manipulations on single person objects	BFS Integrity Signature

Identification	Verification Type	In scope	Out of scope	Specification
DOI_BFS_VERIFY	DOI BFS Integrity	• Detect manipulations on live data view (see following points) • Detect added DOI objects through unauthorized parties • Detect DOI person objects through unauthorized parties • Detect manipulated signatures on single DOI objects	Manipulations on single DOI objects	BFS Integrity Signature

6.2 Verification Requirements

Process Flow	Description	Required Verification Types
Create Filter Version	Before a new version can be created for a particular filter, the integrity of the underlying live data view must be verified. If all person objects and the person BFS integrity checks are successful, the underlying live data view can be considered valid. If any of the verification fails, the user must be informed with an error message and creating a new version must be prevented.	PER_SIG_VERIFY && PER_BFS_VERIFY
Data Export (Version)	Before a version can be exported for a particular filter, the integrity of the underlying version must be verified. If all person objects and the version integrity checks are successful, the linked persons can be considered valid. If any of the verification fails, the user must be informed with an error message and the export must be prevented.	PER_SIG_VERIFY && VER_SIG_VERIFY
Data Export (Filter)	Before a filter can be exported, the integrity of the underlying live data view must be verified. If all person objects and the person BFS integrity checks are successful, the linked persons can be considered valid. If any of the verification fails, the user must be informed with an error message and the export must be prevented.	PER_SIG_VERIFY && PER_BFS_VERIFY

6.3 Verification Error Handling

Every invalid signature validation is logged as a security log type.

6.4 Performance Optimization

To perform signature verification with the highest throughput, the public key stored on the HSM must be exported and cached in-memory. Only the first signature check of the same public key may be read from the HSM, while subsequent checks must use the cached public key for verification.