| Deliverable number | Deliverable title | WP number | Lead Beneficiary | Type | Dissemination level | Due date (in months) |
|---|---|---|---|---|---|---|
| D 7.1 | Interface Specification Document | WP7 | 1- UL TS BV | Report | Public | 7 |
| | | | | | | |

**Author:** **UL TS BV**

**Checked by:** **OTI**

**Submitted by:** **UL TS BV**

**Submitted on:** **08 – 01 - 2016**

Openticketing

# Secure Token Acceptance Sensor

*Behavior and Interface Specification*

| | |
|---|---|
| Version | 1.0 |
| Date | December 11, 2015 |
| Status | final draft |

# Copyright

# Table of Contents

# List of Figures

# List of Tables

# 1. Introduction

## 1.1 European Travellers Club

Open Ticketing Institute has developed a concept for Account Based Ticketing (ABT) that allows public transport schemes to implement local and interoperable ABT services alongside their existing legacy public transport services through provision of a lightweight side-token, a terminal software update and a service-oriented method for connecting ABT fare calculation and payment methods.

Thereby extending the lifetime of their infrastructural investment while at the same time introducing the potential to create and host smart ticketing solutions alongside their existing card based legacy solutions and introduce account based *travelling*: ABT will develop into account-based *travelling* when it is integrated with other mobility services (such as rural transport services, taxi, bike rental and P+R parking), as well as with on-line journey planning/ticketing and real-time journey monitoring. It can also include CRM activities with the customer account and mobile interface being the source and springboard for other advisory and informational based services. All these services will be part of a centralized platform called the European Travellers Club (ETC).

This concept was the basis for the ETC H2020 proposal and introduces the ability for travel schemes to work interoperable through acceptance of each other's public transport cards (with side tokens) at their own terminals. Thereby allowing foreign cardholders, to use their local transport cards, to consume travel services while abroad at participating travel schemes. This proposal was accepted by the European's Horizon 2020 program and started in May 2015. The ETC program will run for 2 years with pilots starting in 2016.

Currently, most transport schemes are card-centric which means that the balance and fare calculation is done on the issued card. In the proposed architecture of ABT all the logic is transferred to the back end, leading to a great simplification of the travel card and the card-acceptance infrastructure. The functionality of the travel document (can now be in different form factors) is merely to identify the user and link the fare to his account in the backend.

In order to test the ABT architecture OTI will launch 3 pilots in Luxembourg, Germany and Holland. The first pilot is to be executed in Luxembourg where travellers can park their car outside the capital city and continue their travel by train. To stimulate travellers to use this service, parking is offered free of charge. More information about this service is presented later in this document.

A terminal in the ABT ID eco-space is either called a sensor or, in this document, a Secure Token Acceptance Sensor (STAS). The purpose of this document is to specify the behaviour and interface of the STAS towards a travel document that has a Generic Secure Token (GST) application, and the communication towards the Hub. In the ABT architecture, the terminal's functionality is mainly to provide a communication channel between the GST application and the back end. This

significantly simplifies the functionality requirements of the terminal as the GST is only used to identify the end user. The STAS's main purpose is to acquire a transaction receipt from the GST and to forward it to the Hub. Additionally, the STAS should be able to verify the receipt's authenticity offline, in case the connection between STAS and backend is (temporarily) unavailable.

## 1.2 Document scope

The scope of this document is:

- The STAS functional requirements
- The byte level layout of the commands between STAS and GST application
- The byte level layout of the commands between STAS and Hub.

## 1.3 Audience

The intended audience for this document is:

- STAS manufacturers
- Security evaluators and external reviewers

## 1.4 Terminology

The words STAS, terminal and sensor are used as synonyms through this document.

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].

## 1.5 List of Acronyms

| Symbol | Meaning |
|---|---|
| AASM | Authorisation and Authentication Software Module |
| ABT | Account Based Ticketing |
| APDU | Application Protocol Data Unit |
| API | Application programming Interface |
| CAL | Card Acceptance List |
| CLA | Class bytes of a command APDU |
| CRM | Customer Relationship Management |
| ECDSA | Elliptic Curve Digital Signature Algorithm |
| ETC | European Travellers Club |
| FCI | File Control Information |
| GST | Generic Secure Token |
| HTD | Host Transaction Data |
| INS | Instruction byte of a command APDU |

| Symbol | Meaning |
|---|---|
| IOTAP | InterOperable Transport Access Provider |
| ISIN | Issuer Specific Identification Number |
| ISO | International Organization for Standardization |
| Lc | Field indicating the length of the data field in a command APDU |
| Le | Field indicating the expected length of the response data |
| MAC | Message Authentication Code |
| P+R | Park and Ride |
| P1 | Parameter 1 of a command APDU |
| P2 | Parameter 2 of a command APDU |
| RFU | Reserved for Future Use |
| STAS | Secure Token Acceptance Device |
| TAL | Terminal acceptance List |
| TSI | Transaction Security Information |
| VDV | Verband Deutsche Verkehrsunternehmen. |

*Table 1: List of acronyms*

## 1.6 Referenced Documents

| Identification | Document |
|---|---|
| ISO 7816-4 | ISO/IEC 7816-4: 2013, Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange, third edition, 15 April 2013 |
| RFC 2119 | Request for Comments: 2119, Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, March 1997 |
| X9.62 | ANSI X9.62-2005, The Elliptic Curve Digital Signature Algorithm (ECDSA), 16 November 2005 |
| RFC5280 | Request for Change: 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, May 2008 |
| RFC5639 | Request for Change: 5639, Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation, March 2010 |

*Table 2: List of referenced documents*

# 2. STAS design

All failures and non-happy flows that are not explicitly specified in this document shall result in a graceful termination of the transaction.

## 2.1 STAS interfaces

In an account-based system the terminal's main purpose is to provide a communication channel between the token application and the back end. To this end the STAS has two interfaces, one for the communication with the GST application and another to communicate with the Hub. The details regarding the communication protocol and the specific commands of each interface are defined in sections 4 and 6 for the GST application and the Hub, respectively.

From the terminal's perspective a transaction is divided into two main steps:

1) The STAS provides information to the GST application and asks it to compile a transaction receipt, which includes a Message Authentication Code (MAC). Optionally the STAS can request the GST application to provide a signature for local authentication.
2) The STAS sends a trigger message to the Hub, which includes the MAC and other transaction-related data.

In the ideal situation, the Hub approves the transaction and the STAS's role is only to communicate the Hub's decision to the end user. However, in some cases the STAS needs to make a business decision before communicating the transaction details to the Hub. This is done by verifying the GST's signature which can be explicitly requested via the Get Transaction Receipt APDU. In the following section we define the transaction flow for Online / Offline Verified transactions.

## 2.2 Sequence diagrams

### 2.2.1 Online

In an online transaction the STAS requests a transaction receipt from the GST application. The transaction receipt includes a cryptogram that can only be verified by the token issuer. The transaction receipt is forwarded to the Hub using a trigger message. Besides the transaction receipt, the trigger message also includes additional transactional and STAS-specific data. The response from the Hub tells the STAS to accept or decline the transaction. The sequence diagram is shown in Figure 1.

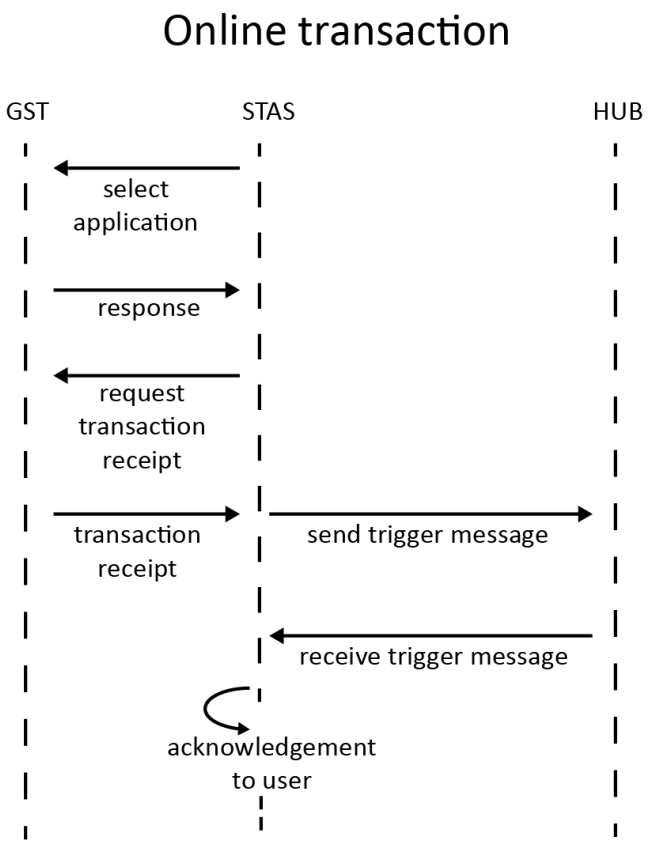



*Figure 1: Online sequence diagram.*

### 2.2.2 Offline verified

For an offline transaction the STAS requests a transaction receipt, similar to the one in the online case. However, in this case the STAS also requests the GST application to provide an additional signature that can be locally verified by the STAS. After the STAS has verified the signature, the STAS performs local risk management and then decideswhether to accept or deny the transaction. If the transaction is approved then an acknowledgment is sent to the user. In both situations (approve/ deny) a trigger message is sent to the Hub for further processing. The transaction flow is shown in Figure 2.

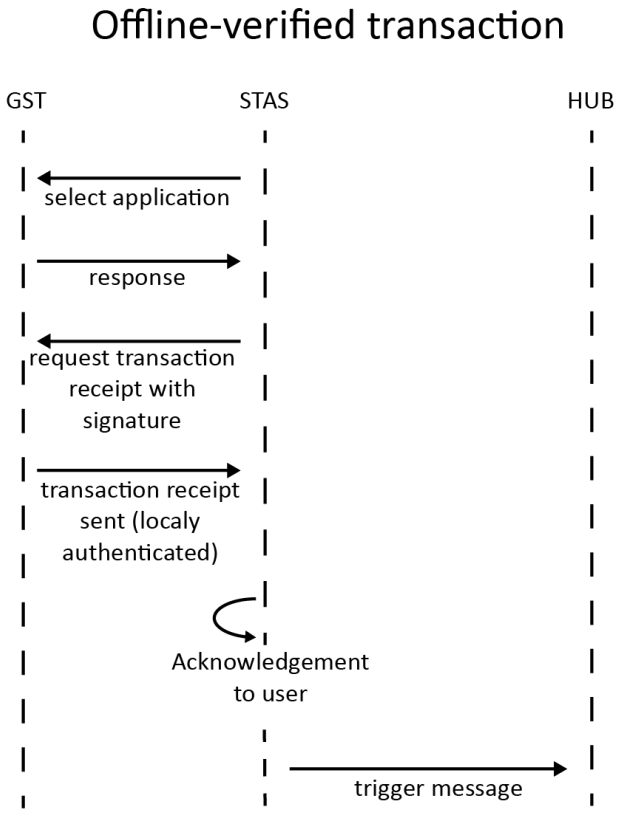



*Figure 2: Offline verified transaction flow*

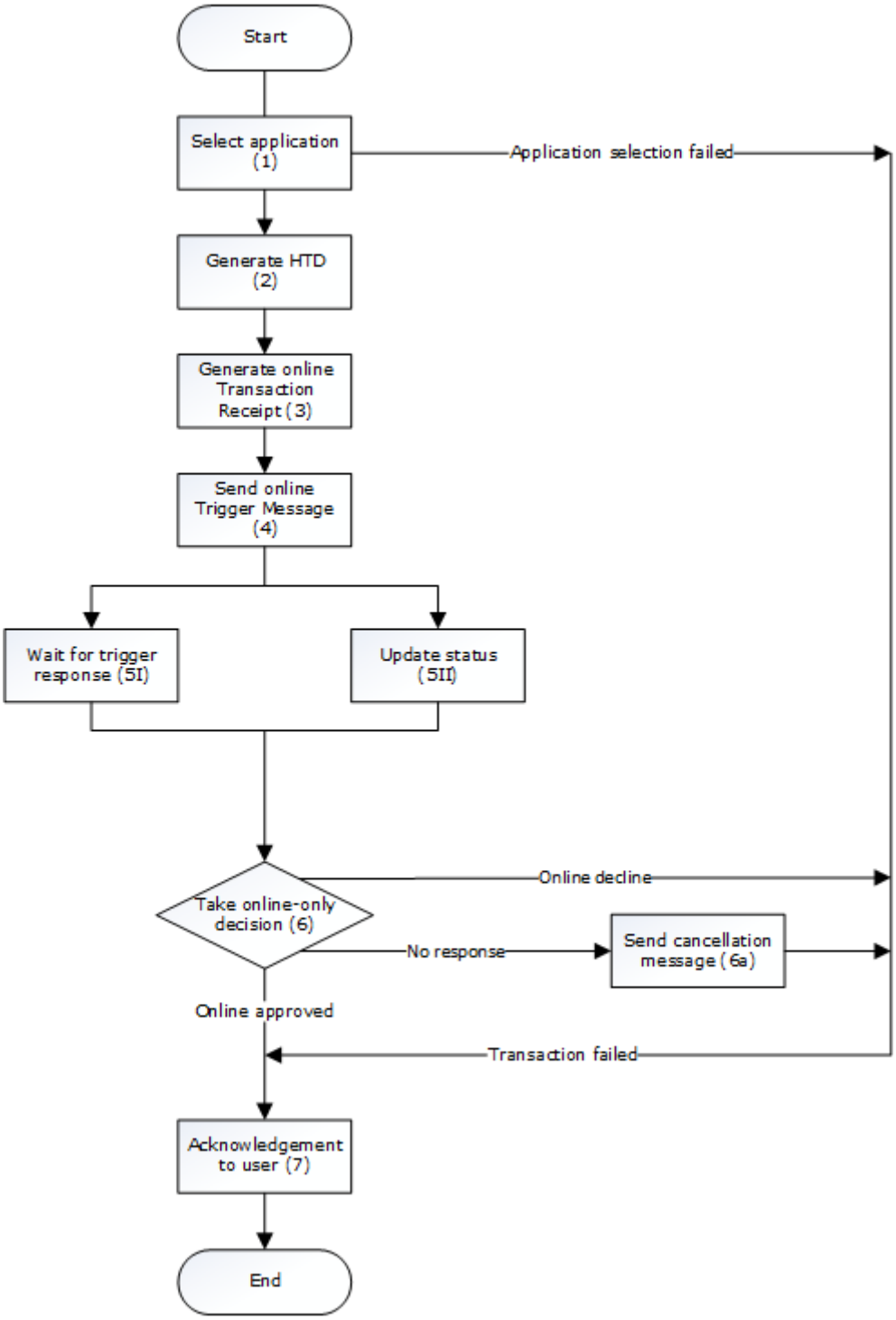
## 2.3 STAS operation modes

Depending on the specific implementation, the STAS can operate in one of four operation modes. In the following sub-sections we define the steps taken by the STAS in each of these operation modes, including pseudo-code to communicate with the GST and Hub interfaces.

### 2.3.1 Online only

In the online-only mode, only the HUB can authorize the transaction. The STAS must therefore wait for a response from the Hub, or decline the transaction if the response times out.

Flow diagram:




Main Success Scenario:

| Steps | Action | comments |
|---|---|---|
| 1 | *Select GST application (§3.1)* | |
| 2 | *Generate HTD (§3.2)* | |
| 3 | *Generate transaction receipt (§3.3)* | *For online authentication* |
| 4 | *Send trigger message (§3.4)* | *For online approval* |

| Steps | Action | comments |
|---|---|---|
| 5I | *Wait for trigger response (§3.5)* | *For the pre-set timeout time* |
| 5II | *Update status information (§3.6)* | |
| 6 | *Take online-only decision (§3.9)* | |
| 7 | *Acknowledgement to user (XXX)* | |

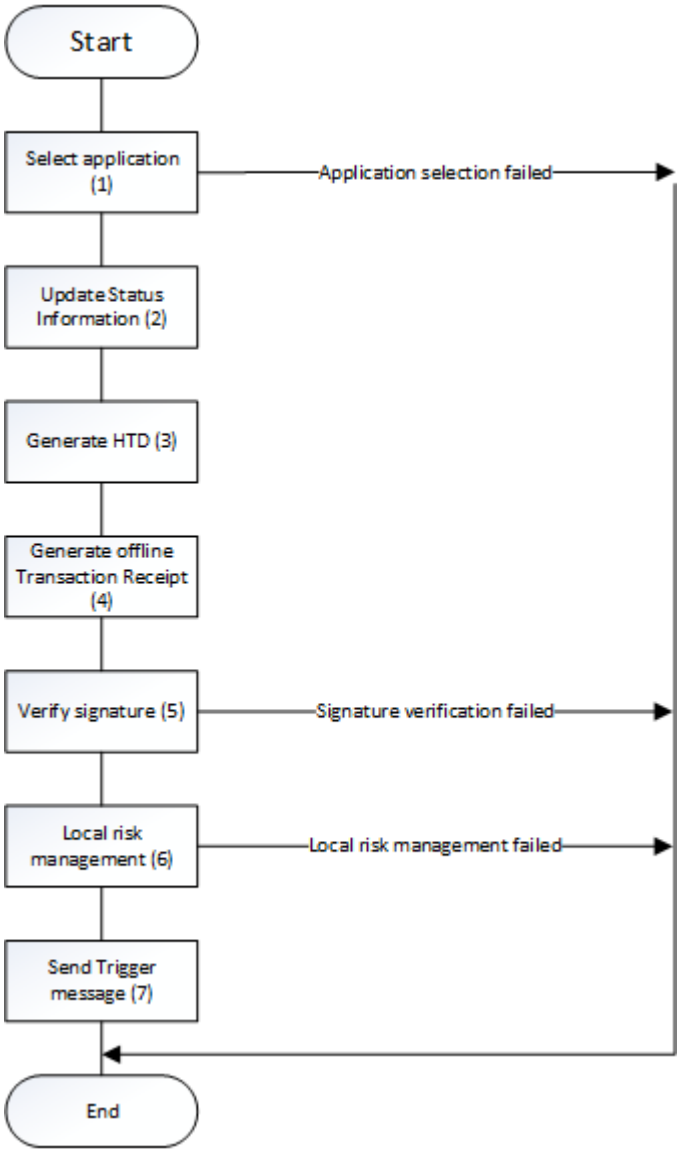
Extensions:

| Steps | Actions |
|---|---|
| 1a | GST application not selected<br>*End in the 'Transaction failed' state* |
| 6a | Online decline, send cancellation message<br>*End in the 'Transaction failed' state* |
| 6b | Timeout<br>*End in the 'Transaction failed' state* |

### 2.3.2 Autonomous (offline)

In this mode, the STAS has the capability to locally authenticate the GST. After signature verification and dependent on deterministic local risk management, the STAS approves or declines the transaction. A trigger message is sent to the Hub for further processing rather than for approval.

Flow diagram:




Main Success Scenario:

| Steps | Action | comments |
|---|---|---|
| 1 | *Select GST application (§3.1)* | |
| 2 | *Update Status Information (§3.6)* | |
| 3 | *Generate HTD (§3.2)* | |
| 4 | *Generate transaction receipt (§3.3)* | *With additional signature* |
| 5 | *Verify signature (§3.7)* | |
| 6 | *Local risk management (§3.8)* | |
| 7 | *Send trigger message (3.4)* | *Store and forward* |

Extensions:

| Steps | Actions |
|---|---|
| 1a | GST application not selected |

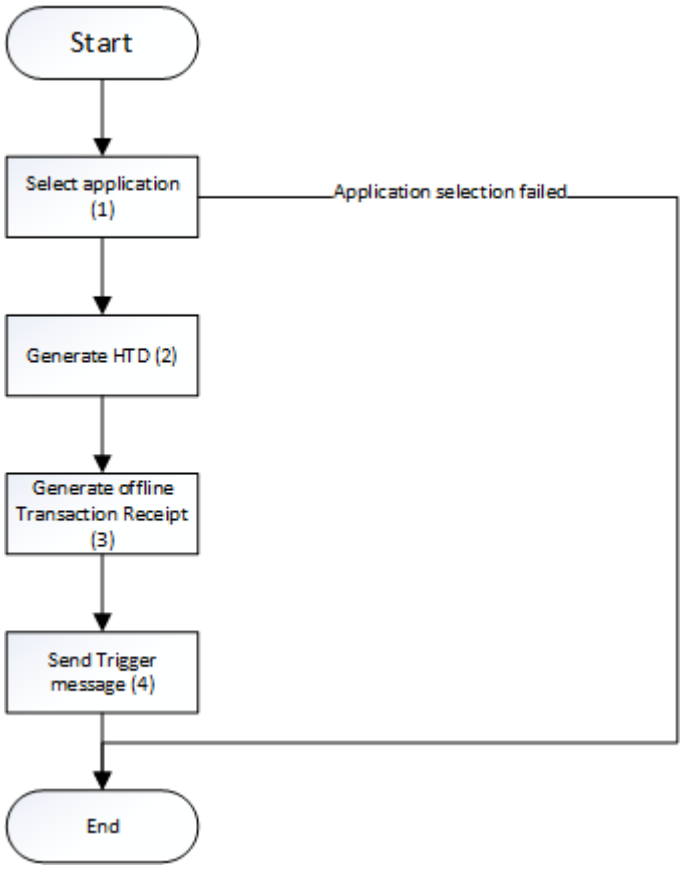
| | |
|---|---|
| | *End in the 'Transaction failed' state* |
| 5a | Signature verification failed<br>*End in the 'Transaction failed' state* |
| 6a | Local risk management failed<br>*End in the 'Transaction failed' state* |

### 2.3.3 Autonomous not verified

In this mode the STAS requests a transaction receipt and sends a Trigger message to the Hub. The STAS does not need to make a decision whether to approve or deny. Furthermore, no signature verification or local risk management is required.

This mode of operation is only to be used in situations where the transaction between GST and STAS does not directly lead to business relevant impact and thus does not have direct value. This is for example the case if transactions are performed only as a proof-of-presence.

Flow diagram:




Main Success Scenario:

| Steps | Action | comments |
|---|---|---|
| 1 | *Select GST application (§3.1)* | |
| 2 | *Generate HTD (§3.2)* | |

| Steps | Action | comments |
|---|---|---|
| 3 | *Generate transaction receipt (§3.3)* | *Without additional signature* |
| 4 | *Send trigger message (§3.4)* | *Store and forward* |

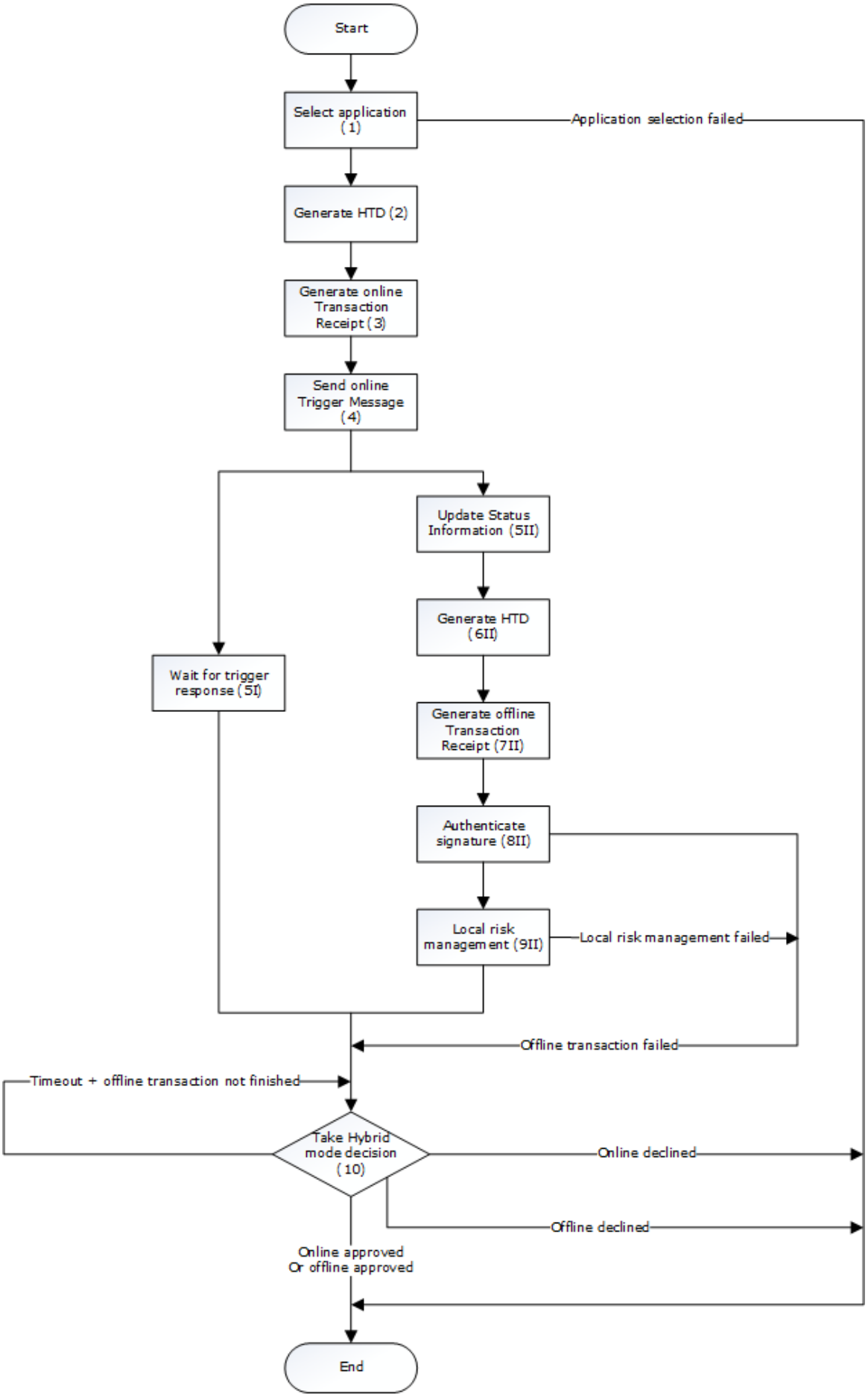
Extensions:

| Steps | Actions |
|---|---|
| 1a | GST application not selected<br>*End in the 'Transaction failed' state* |

### 2.3.4 Hybrid mode

In this operation mode the STAS will first send a trigger message for online approval from the Hub. While waiting for a response the STAS will ask for an additional transaction receipt that can be locally verified. If a response from the Hub arrives within the "online timeout" then the STAS will respond according to the Hub's decision. If no response from the Hub arrives on time, then the STAS will approve/deny the transaction according to the local signature verification and local risk management.

Flow diagram:


Start
Select application (1)
Application selection failed
Generate HTD (2)
Generate online Transaction Receipt (3)
Send online Trigger Message (4)
Update Status Information (5II)
Generate HTD (6II)
Wait for trigger response (5I)
Generate offline Transaction Receipt (7II)
Authenticate signature (8II)
Local risk management (9II)
Local risk management failed
Offline transaction failed
Timeout + offline transaction not finished
Take Hybrid mode decision (10)
Online declined
Offline declined
Online approved
Or offline approved
End

Main Success Scenario:

| Steps | Action | Comments |
|---|---|---|
| 1 | *Select GST application (§3.1)* | |
| 2 | *Generate HTD (§3.2)* | |
| 3 | *Generate transaction receipt (§3.3)* | *For online authentication* |
| 4 | *Send trigger message (§3.4)* | *For online approval* |
| 5I | *Wait for trigger response (§3.5)* | *For the pre-set timeout time* |
| 5II | *Update status information (§3.6)* | |
| 6II | *Generate HTD (§3.2)* | |
| 7II | *Generate transaction receipt (§3.3)* | *with additional signature* |
| 8II | *Authenticate signature (§3.7)* | |
| 9II | *Local risk management (§3.8)* | |
| 10 | *Hybrid mode decision (§3.10)* | |

Extensions:

| Steps | Actions |
|---|---|
| 1a | GST application not selected<br>*End in the 'Transaction failed' state* |
| (8II)a | Signature verification failed<br>*Go to step 10 with status 'offline transaction failed'* |
| (9II)a | Local risk management failed<br>*Go to step 10 with status 'offline transaction failed'* |
| 10a | Transaction declined by Hub OR (No response from Hub AND offline transaction failed)<br>*End in the 'Transaction failed' state* |

## 2.4 STAS onboarding

### 2.4.1 STAS configuration

Before the STAS can operate within the ETC platform it needs to be configured with initial data. All other data is optional, service context specific or can be generated on the fly:

| Source | Dest. | Data | Reference | M/O |
|---|---|---|---|---|
| Terminal | Hub | ISIN_STAS | 2.4.1.1 | M |
| Terminal | Hub | Operation mode | 2.4.1.2 | M |

| | | | | |
|---|---|---|---|---|
| ETC | Terminal | ETC CA public key | 3.7.2 | O[1] |
| Hub | Terminal | Hub public key | 6.5 | O[2] |
| Hub | Terminal | ServiceID | | M |
| | Terminal | Local Risk Management parameters | 3.8.5 | O[3] |
| Terminal | Hub | Timeout period | 2.4.1.5 | O[4] |
| Terminal | Hub | Supported Tokens | | M |
| Hub | Terminal | Web service endpoint URL | | M |
| | | SensorID | 2.4.1.3 | M |
| Hub | Terminal | Supported Token Issuers | 3.8.4 | M |
| | Terminal | Counter | 0 | M |
| Terminal | Hub | Salt | 6.2 | M |

Note: Pre-configuration is a manual process, please contact ETC for pre-configuration of test and pilot STASs.

#### *2.4.1.1 ISIN_STAS*

ISIN_STAS (4 bytes) is set by the STAS vendor and shared with the Hub. ISIN_STAS consists of Manufacturer byte (assigned by ETC) + 3 bytes that may be derived from the SensorId (e.g. last 3 bytes of SensorId) or from manufacturer-specific identifier (Device ID, Terminal ID etc). Must be communicated at onboarding of sensor.

#### *2.4.1.2 Operation mode*

The operation mode shall be determined by the service provider / terminal manufacturer and shall be either:

1. Online
2. Autonomous Verified
3. Autonomous Not-Verified
4. Hybrid

#### *2.4.1.3 SensorID*

The SensorId (16 byte GUID) is issued either by the STAS manufacturer or by the local Hub

[1] In case of Hybrid Mode or Autonomous Verified Mode

[2] In case of Hybrid Mode, Online Mode and if Heartbeats are supported

[3] IN case of Hybrid Mode and Autonomous Verified Mode

[4] In case of Online and Hybrid Mode

#### 2.4.1.4 Counter

The Counter is a 3 byte binary number that shall be used by the STAS for all transactions and Trigger Messages

The STAS SHALL persistently store the Counter. The initial value of the Counter, before the first execution of the Get Transaction Receipt command or the first Trigger message, SHALL be set to 0x000000.

The STAS implementation SHALL guarantee that the same Counter value is not used for more than one transaction.

The STAS SHALL therefore increment the Counter by 1 or more and persistently store the new value of the Counter before the STAS includes the Counter in the command data field of the Get Transaction Receipt command. In case the Counter has reached the value 0xFFFFFF, the STAS SHALL no longer perform any Get Transaction Receipt commands and the Transaction Sequence Number SHALL NOT be rolled over to the value 0x000000.

#### 2.4.1.5 Timeout Period

A STAS with any of the Operation modes Online and Hybrid SHALL be configured with a time-out parameter (in milliseconds) by the Service Provider or Terminal Manufacturer. This parameter must be shared with the HUB.

# 3. Sub functions

## 3.1 Select GST application

To select the GST Application the STAS SHALL execute the following steps in the order listed:

1) PICC activation according to ISO 14443-3
2) Protocol activation according to ISO 14443-4
3) GST application selection according to ISO 7816-4

The STAS SHALL select the Generic Secure Token Application in accordance with ISO 14443-3 and ISO 14443-4. In particular for ISO 14443-3 Type A the STAS SHALL verify that the SAK byte indicates support for ISO 14443-4. The STAS SHALL only continue with 14443-4 protocol activation if bit b6 of the SAK byte has value 1 (refer to ISO 14443-3).

The STAS SHALL perform GST Application selection according to [ISO7816-4, § 8.2.2.2] where the APDU shall be formatted according to section 4.2.

For the application selection the STAS SHALL use the (truncated) AID parameter value:

Truncated AID: 0xA0 00 00 05 93 2E 01

The FCI data responded by the GST application upon selection is:

| Tag | Length | Value (description) | | |
|---|---|---|---|---|
| 0x6F | var. | FCI-template | | |
| | **Tag** | **Length** | **Value (description)** | |
| | 0x84 | 0x05..<br>0x10 | DF-name<br>(Full AID of the selected application.) | |
| | 0xA5 | 0x15 | FCI-proprietary template | |
| | | **Tag** | **Length** | **Value (description)** |
| | | 0x41 | 0x10 | TokenId (10 bytes 20character BCD) |
| | | 0x9F7D | 0x02 | Build number |

The STAS SHALL perform the following verifications on the response to the Select command in the order listed.

1. The STAS SHALL verify that the response to the application selection contains FCI data as indicated by tag 0x6F ([ISO7816-4, § 5.3.3]). If the response does not contain tag 0x6F the STAS SHALL abort further processing.
2. The STAS SHALL verify that the full AID as present in tag 0x84 ([ISO7816-4, § 5.3.3]) of the FCI data returned in response to the application selection is supported by the STAS. If the full AID is not within the set of Generic Secure Token AID values supported by the STAS, the STAS SHALL abort further processing.

The current set of full AIDs for to be supported with this STAS specification is:

0xA0 00 00 05 93 2E 01 02 10

## 3.2 Generate HTD

The Host Transaction Data (HTD) is a field generated by the STAS which is given to the GST application as an input in the Get Transaction Receipt APDU. The HTD will be a hash of the Trigger Message fields that are set by the STAS, in this way the integrity of the Trigger Message is included in the Transaction MAC.

Prior to the construction of the HTD, the STAS SHALL increase the Counter variable with one (1).

The input to the HTD comprises the Trigger Message Data Objects (§6.4.2)

- Transaction (§6.4.2.1)
- Sensor (§6.4.2.4)
- Service (§6.4.2.7)
- ServiceRequestData (§6.4.2.8)

The computation of the HTD consists of two steps:

1. Concatenate all the values in the Data Objects listed above in the order specified in (§6.4.2). If PropertyBags are used, then the order used to construct the HTD will determine the order used to construct the Trigger Message.
2. HTD is a SHA-256 hash that is calculated over the result of step 1.

The values to construct the HTD are stored and subsequently used for sending the trigger message.

## 3.3 Generate transaction receipt

The STAS SHALL send a Get Transaction Receipt APDU (§4.3) to the GST application. The P1 field is set to 0x00 for an online receipt and 0x01 for an offline-verifiable receipt.

The response to this APDU (defined in Table 4) consists of parameters that SHALL be used in the trigger message.

The STAS SHALL verify that the SW1/SW2 field in the response APDU is equal to 0x90 0x00, otherwise the transaction is terminated.

## 3.4 Send trigger message

### 3.4.1 Assembling the TSI

The STAS SHALL extract from the Get Transaction Receipt response the 8 byte TSI_GST field. The STAS SHALL concatenate, in this order, the fields in the table below:

| Field | Length (bytes) | Description |
|---|---|---|
| TSI_GST | 8 | GST part of the Transaction Security Information (Table 4) |

| | | |
|---|---|---|
| Status Information | 8 | Status information as stored in the GST (Table 4) |
| ISIN_STAS | 4 | Issuer Specific Identification Number of the STAS |
| Counter | 3 | Counter |

### 3.4.2 Composing Trigger Message

Using the response to the Get Transaction Receipt APDU and the values used in §3.2, the trigger message can be sent (according the Hub API in 6.4). The response, the HTD and the TSI shall be used to fill the Tokens object as follows:

```
TriggerMessage::Tokens[1].TokenType = "GST"

TriggerMessage::Tokens[1].TokenValue = GetTransactionReceipt.response.TokenID

TriggerMessage::Tokens[1].Propertybag[1]=("HTD",HTD)

TriggerMessage::Tokens[1].Propertybag[2] =
("GSTversion", GetTransactionReceipt.response.GSTversion)

TriggerMessage::Tokens[1].Propertybag[3] = ("TSI",TSI)

TriggerMessage::Tokens[1].Propertybag[4] =
("TMAC", GetTransactionReceipt.response.TransactionMAC)
```

After the Trigger Message is sent to the Hub a reference (TransactionID) to the message is kept locally until a response from the Hub is received.

Addition: In the Hybrid mode, the STAS SHALL add the following field to the transaction and offline trigger message:

```
TriggerMessage::Transaction.ReferencedTransaction = (previous online TransactionID)
```

## 3.5 Wait for trigger response

After a trigger message has been sent to the Hub, where the message RequestMode was 1 (online) the STAS needs to wait for approval / denial of the transaction. The STAS waits for a maximum duration set by the parameter Online_timeout.

Trigger message responses that are received during this mode need to be checked whether the response is related to the message waiting for approval (it can also be a response to a message sent earlier which was delayed).

IF the response is to the online pending trigger message

THEN

- Go to the next step.

ELSE

- Process the response and continue to wait for the trigger response.

This process is finished with one of the following results:

1. Approved: positive response from the Hub

2. Denied: negative response from the Hub
3. Timeout: no response from the Hub in time

## 3.6 Update status information

The STAS holds an action list that the STAS must refresh periodically (§6.3). The TokenIDs on this list (TokenHash) are hashed according to section 6.2. The STAS SHALL verify if the TokenID is on the ActionList. The STAS may use the TokenID in the FCI (see §3.1) for performance purposes. If the TokenID is on the ActionList (§6.3.2.1), the STAS shall send the APDU as contained in the associated field "APDUValue" (§6.3.2.2) to the GST.

The STAS SHALL disregard the response to the APDU and continue processing.

## 3.7 Verify signature

The information needed to verify the signature is divided into two parts. First we explain how to retrieve the token's public key. Then we explain how the signature is verified.

### 3.7.1 Retrieval of the token's public key

The token's public key can be found in the end-entity certificate. The STAS SHALL get this certificate from the GST application using the Get Certificate command as defined in §4.4.

In order to check whether the token's public key is genuine, the signature of the end-entity certificate needs to be verified. This is done using the issuer's public key that is stored in the subordinate certificate of the token issuer. There are two options:

1) If the subordinate certificate of the token is cached in the STAS (the link between the certificates is made with keyIdentifier value in the AuthorityKeyIdentifier field in the subordinate and end-entity certificates, for more information see RFC5280) then the issuer's public key is already known and can be used to authenticate the signature of the end entity certificate.
2) If the relevant subordinate certificate is not cached THEN
    1) Use the Get Certificate command (as defined in §4.4) to retrieve the subordinate certificate from the GST application.
    2) Use the ETC public key (pre-loaded in the STAS) to verify the signature of the subordinate certificate. If this succeeds, then cache the subordinate certificate for future transactions.
    3) Use the issuer's public key (from the Subordinate certificate) to authenticate the end entity certificate.
    4) If all the above-mentioned steps are successful, then the token's public key is genuine. Otherwise, abort the transaction.

### 3.7.2 Certificate and signature verification

Refer to Appendix A. Certificate profiles for an overview of the profiles of the GST end-entity certificate and the Subordinate certificate

#### *3.7.2.1 Subordinate certificate*

The Subordinate certificate shall be verified as follows:

1. The ECDSA signature over the Subordinate public key SHALL be verified according to [X9.62, §7.4], using SHA256 as the hash function and elliptic curve domain parameters as specified by [RFC5639, § 3.3] with fixed Curve-ID brainpoolP256r1 and with the Root-CA public key.
2. The validity of the certificate shall be verified
3. The Organizational Unit Name shall be verified to coincide with the type of environment the STAS is in (Development, Test, Acceptance or Production). In a production environment, only Production Environment certificates shall be accepted

#### *3.7.2.2 Token certificate*

The Token certificate shall be verified as follows:

1. The ECDSA signature over the Subordinate public key SHALL be verified according to [X9.62, §7.4], using SHA224 as the hash function and elliptic curve domain parameters as specified by [RFC5639, § 3.3] with fixed Curve-ID brainpoolP224r1 and with the Subordinate public key.
2. The validity of the certificate shall be verified
3. The Organizational Unit Name shall be verified to coincide with the type of environment the STAS is in (Development, Test, Acceptance or Production). In a production environment, only Production Environment certificates shall be accepted
4. The Common Name shall be verified to be the same as the TokenID of the corresponding GST.

#### *3.7.2.3 Token Signature*

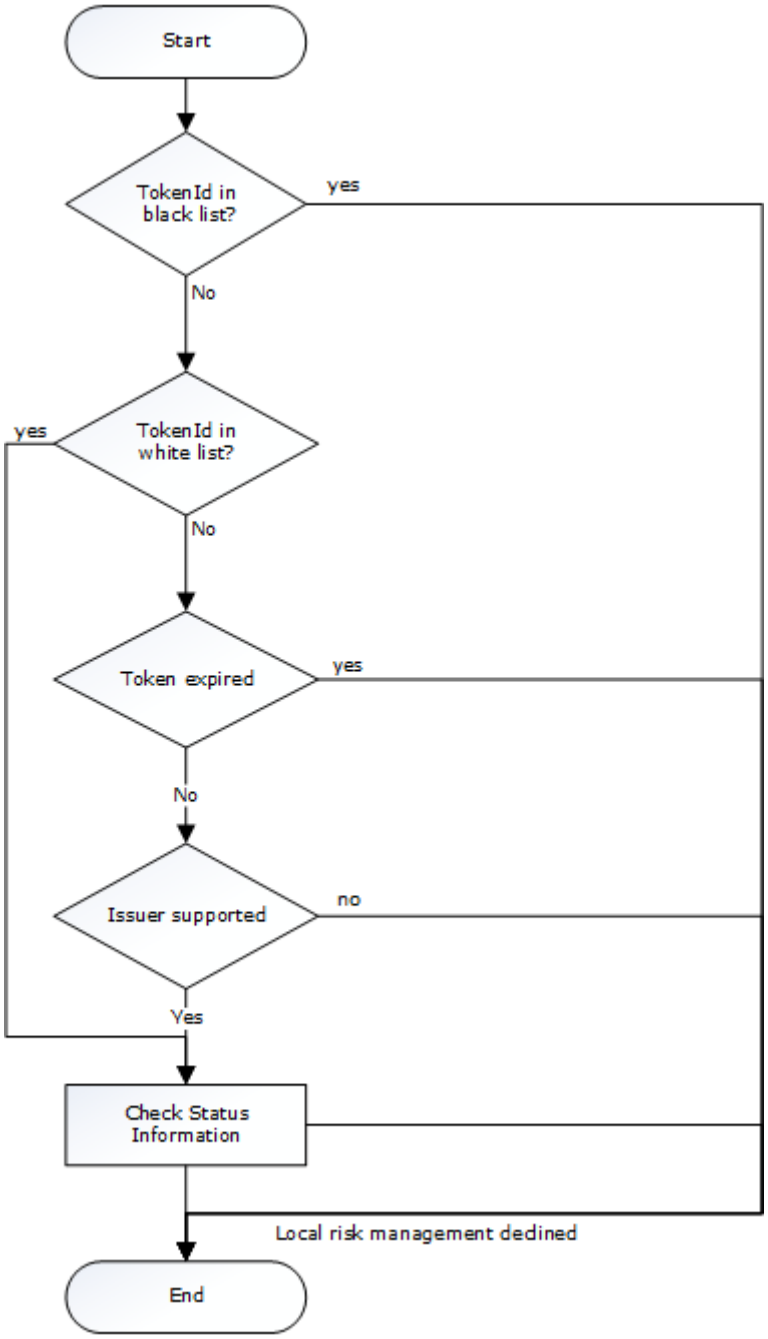
The data that is signed by the GST comprises the concatenation of the fields "TokenID" up to and including "Transaction MAC" of the response APDU formatted as per Table 4. The signature itself consists of the fields Signature_r and Signature_s according to the same Table 4.

The ECDSA signature SHALL be verified according to [X9.62, §7.4], using SHA224 as the hash function and elliptic curve domain parameters as specified by [RFC5639, § 3.3] with fixed Curve-ID brainpoolP224r1

If the signature consists of zero bytes (0x00), the STAS shall decline the transaction.

## 3.8 Local risk management

The local risk management is composed of 5 parts: a black list, a white list, token expiration date check, list of supported issuers and the token's Status Information check. The process of local risk management is illustrated in the following flow chart.




### 3.8.1 Black list

If the token is on the black list then the local risk management is set to "failed" and the rest of the steps of the risk management are skipped.

### 3.8.2 White list

If the token is on the white list then the local risk management is set to "passed" and the STAS shall continue with a Check of the Status Information as per §3.8.5.

### 3.8.3 End Date

The STAS SHALL verify whether the End Date of the GST (see Table 4) is after the current date.

If this is not the case, the STAS shall decline the transactions

### 3.8.4 Supported Token Issuer

The STAS SHALL verify whether the first four (4) digits TokenID (most significant 2 bytes in BCD) are on the list of Supported Token Issuers. If this is not the case, the STAS shall decline the transaction.

### 3.8.5 Status Information

The status information is an 8 byte field in the GST application that can be changed dynamically by the token issuer. The status information is defined as follows:

| Byte 8 | Byte 7 | Byte 6 | Byte 5 | Byte 4 | Byte 3 | Byte 2 | Byte 1 |
|---|---|---|---|---|---|---|---|
| GAL | | | | | | | GVAL |

*Table 1: Status information*

Byte 1 represents the maximum offline authorized travel amount. Bytes 2 to 8 is a list (Card Acceptance List, CAL) representing business and risk rules.

The STAS SHALL have the following Terminal Risk Parameters:

| Byte 8 | Byte 7 | Byte 6 | Byte 5 | Byte 4 | Byte 3 | Byte 2 | Byte 1 |
|---|---|---|---|---|---|---|---|
| SAL | | | | | | | SVAL |

GAL is the GST Acceptance List and SAL the Sensor Acceptance List.

GVAL is the GST Value and SVAL the Sensor Value.

For the status information risk management to pass both following conditions have to be met:

1) The GVAL field in the GST application must be higher or equal to the value of SVAL in the Terminal Risk Parameters.
2) A bitwise AND operation between GAL and SAL must be equal to the SAL.

In all other cases the STAS SHALL decline the transaction.

## 3.9 Online-only decision

This process starts after inputs from 'wait for trigger response' and 'update status' have arrived.

There are 3 possible scenarios:

Option 1: A response to the trigger message is positive

Action: Acknowledge approval to user (gate open, display message, etc)

Option 2: A response to the trigger message is negative

Action: Acknowledge denial to user

Option 3: No response has arrived on time

Action: send a cancellation message to the Hub (tbd) and abort the transaction.

## 3.10 Hybrid-mode decision

This process starts after the input from the online thread has arrived.

According to the result of the "wait for trigger response" (§3.5):

**Approved:**

- Terminate the offline thread in a graceful manner.
- Acknowledge approval to user (open gate, present text, etc)

**Denied:**

- Terminate the offline thread
- Terminate the transaction
- Acknowledge denial to user (present text, etc.)

**Timeout:**

- If the offline thread has already executed the local risk management THEN
    - If the local risk management failed then acknowledge to the user, send a cancellation message (tbd) and terminate the transaction.
    - If the local risk management is successful then send a "store and forward" trigger message referencing to the previous timed-out transaction (see §3.4) and acknowledge to the user.

  ELSE (online timeout but the offline thread not finished yet, unlikely to happen)
    - Wait for trigger response OR the local risk management is executed. Proceed with whichever arrives first.

# 4. GST Interface

## 4.1 Physical layer

The GST Application is intended to be hosted on a platform that fully supports the protocol according to ISO/IEC 14443-4, commonly referred to as T = CL.

## 4.2 Select Application

**Command**

| Field | Value | Description |
|---|---|---|
| CLA | 0x00 | Inter Industry Class |
| INS | 0xA4 | SELECT instruction |
| P1 | 0x04 | By DF name / Application ID |
| P2 | 0x00 | First or only occurrence and returning the FCI template |
| Lc | 0x0? | Length of truncated AID in command data |
| Data | <…> | Truncated AID of GST Application |
| Le | 0x00 | Length expected |

**Response**

| Field | Length (byte) | Description |
|---|---|---|
| FCI data | Variable | FCI data according to [ISO7816-4, § 5.3.3] |
| SW1/SW2 | 2 | 0x90 0x00, status word: Ok |

## 4.3 Get Transaction Receipt

**Command**

| Field | Value | Description |
|---|---|---|
| CLA | 0x80 | Proprietary class ([ISO7816-4, § 5.1.1]) |
| INS | 0xFA | Get Transaction Receipt |
| P1 | 0x00<br>0x01 | Determines whether a signature is to be generated for offline verification. Supported values:<br>0x00: No signature is requested<br>0x01: A signature is requested for offline verification<br>0x02..0xFF: not applicable |
| P2 | 0x00 | Value is not guaranteed to be verified by GST Application. |

| Field | Value | Description |
|---|---|---|
| Lc | 0x27 | Length of command data |
| Data | <…> | See Table 3: Get Transaction Receipt Data fields |
| Le | 0x00 | Variable length of response |

*Table 2: Get Transaction Receipt Command fields*

**Data**

| Name | Length (byte) | Description |
|---|---|---|
| ISIN_STAS | 4 | Host (STAS) Identifier |
| Counter | 3 | STAS transaction counter |
| HTD | 32 | HostTransactionData |

*Table 3: Get Transaction Receipt Data fields*

**Response**

| Name | Length (byte) | Description |
|---|---|---|
| TokenID | 10 | Token Identifier, 10 bytes representing a 20 digit number in BCD encoding. |
| Enddate | 4 | Token End date, signed integer representing the number of seconds elapsed since midnight (UTC) 1-1-1970. |
| GSTversion | 2 | Application Version |
| TSI_GST | 8 | Transaction Security Information from the GST |
| StatusInformation | 8 | Value of the Status Information as stored persistently on the GST. |
| TransactionMAC | 10 | GST MAC over the transaction between STAS and GST. The value of this TMAC is verified by the Hub. |
| Signature_r | 0 or 28 | This field is present if and only if a signature was requested for offline verification. The value of the field shall be part r of the signature, pre-padded with zeros to obtain a field length of 28 byte. |
| Signature_s | 0 or 28 | This field is present if and only if a signature was requested for offline verification. The value of the field shall be part s of the signature, pre-padded with zeros to obtain a field length of 28 byte. |
| SW1/SW2 | 2 | 0x90 0x00: Command OK |

*Table 4: Get Transaction Receipt response.*

**Command specific extra response codes**

| SW1 | SW2 | Description |
|---|---|---|
| 0x69 | 0x86 | Authentication method blocked. Card sequence counter reached maximum. |
| 0x67 | 0x00 | Wrong value for Le or Lc |

*Table 5: Get Transaction Receipt extra response codes.*

## 4.4 Get Certificate

**Command**

| Field | Value | Description |
|---|---|---|
| CLA | 0x80 | Proprietary class ([ISO7816-4, § 5.1.1]) |
| INS | 0xCA | Get Certificate |
| P1 | 0x00<br>0x01 | Indication of the certificate type:<br>0x00 = GST End Entity Certificate<br>0x01 = CA Subordinate Certificate |
| P2 | 0x00<br>0x01 | Signals whether this is the first or a subsequent Get Certificate command:<br>0x00: first command in sequence<br>0x01: next command, used to retrieve remaining certificate data |
| Lc | - | Not present |
| Data | - | Not present |
| Le | 0x00 | Variable length of response |

*Table 6: Get Certificate Command fields.*

**Response**

| Name | Length (byte) | Description |
|---|---|---|
| Certificate Data | variable | Part of the certificate selected by P1. |
| SW1/SW2 | 2 | 0x90 0x00: Complete certificate has been returned<br>0x9F 0xXX: 0xXX additional bytes available for retrieval by next Get Certificate command with P2 set to 0x01. (0xXX = 0x00 indicates 256 or more bytes available). |

*Table 7: Get Certificate Response.*

**Command specific extra response codes**

| SW1 | SW2 | Description |
|---|---|---|

| | | |
|---|---|---|
| 0x69 | 0x86 | Get Certificate command not allowed. In this case P2 = 0x01 but there is no directly preceding APDU with P2 = 0x00. |

*Table 8: Get Certificate extra response codes.*

# 5. Data Types

## 5.1 Data Types

For the Hub interface, the JSON specific datatypes are referred to in the tables in §6. The byte layout for the GST interface is provided in section 4 above. The mapping between the two interfaces is provided in this section.

## 5.2 Transaction and Trigger Message

The following table shows how values must be encoded on the different interfaces during a transaction with a GST. Examples are provided for those values that may be ambiguous.

| Data Field | GST | Input to HTD | HUB API |
|---|---|---|---|
| TransactionID | - | UTF-8 "yyyyMMddHHmmssfff" | UTF-8 "yyyyMMddHHmmssfff" |
| Example | | `20151210191159000` | `"20151210191159000"` |
| Counter | Binary unsigned integer | - | UTF-8 |
| Example | `0x0000A0` | | `160` |
| SensorId | - | UTF-8 | UTF-8 |
| Example | | `f9af65da-28ad-4a34-9ad5-947681f74307` | `"f9af65da-28ad-4a34-9ad5-947681f74307"` |
| ReferencedTransaction | - | UTF-8 | UTF-8 |
| Example | | `20151210191159000` | `"20151210191159000"` |
| ExternalTransactionID | - | UTF-8 | UTF-8 |
| IdentifierType | - | UTF-8 | UTF-8 |

| | | | |
|---|---|---|---|
| IdentifierValue | - | UTF-8 | UTF-8 |
| TokenType | - | UTF-8 | UTF-8 |
| Example | | GST | "GST" |
| TokenValue / TokenID | 20 digit number in 10 byte BCD | - | UTF-8 |
| Example | 0x00102030405060708090 | | "00102030405060708090" |
| Token.Propertybag.Values | Binary | - | Base64 encoded binary in UTF-8 |
| Example | 0xA0102030405060708 09F | | "QTAxMDIwMzA0MDUwNjA3MDgwOUY=" |
| Service ID | - | UTF-8 | UTF-8 |
| Example | | 8 | 8 |
| RequestInternalIpAddress | - | UTF-8 | UTF-8 |
| Example | | 192.168.1.255 | "192.168.1.255" |
| RequestExternalIpAddress | - | UTF-8 | UTF-8 |
| Example | | 74.125.224.72 | "74.125.224.72" |
| RequestSensorLocalTimestamp | | UTF-8 | UTF-8 |
| Example | | 20151210191159000 | "20151210191159000" |
| Amount | | Binary unsigned integer | UTF-8 |
| Example | | 0x0512 | 1298 |
| CurrencyCode | | UTF-8 | UTF-8 |

| | | | |
|---|---|---|---|
| Example | | EUR | "EUR" |
| RequestMode | | UTF-8 | UTF-8 |
| Example | | 1 | 1 |
| OfflineResult | | UTF-8 | UTF-8 |
| Example | | 1 | 1 |

## 5.3 Retrieve List

The following table shows how values should be interpreted on the different interfaces for list management. Examples are provided for those values that may be ambiguous.

| Data Field | GST | HUB API |
|---|---|---|
| TokenHash | Binary | Base64 encoded hash of the UTF-8 TokenId |
| Example | 0x00102030405060708090 | 813d1ffa03198ad7a8880dc805cb363b81ba7197e42527f1d62e615d50997d4e |
| APDUValue | Binary | Base64 encoded binary data in UTF-8 |
| Example | 0x80A8000000000000000000010001A2B3813d1ffa03198ad7 | ODBBODAwMDAwMDAwMDAwMDAwMDAxMDAwMUEyQjM4MTNkMWZmYTAzMTk4YWQ3 |

# 6. Hub Interface

## 6.1 Physical layer

The Hub API supports message exchange via a HTTP(S) REST interface currently supporting XML & JSON message structures. Due to the message size overhead of XML, JSON format is recommended.

## 6.2 Token Hashing

The Token IDs in a White, Black or Action List are never transferred in the clear. They are salted and hashed and the STAS must run the Token ID it sees through the same process to match a GST to a hashed list.

The hashed TokenID is the result of a hashing operation on the TokenID concatenated with a salt. The hashing algorithm is SHA256.

Hashed TokenID = SHA256 (TokenID + Salt)

The value of the salt is to be agreed between the sensor manufacturer and the Travel Scheme during the onboarding of Sensors. The salt can be static per Travel Scheme but can also be different per Sensor – e.g. derived from a sensor identifier - or per sensor manufacturer.

## 6.3 Retrieve Aggregate List

Retrieves a list of ETC Token IDs that are white or black listed listed for a given service hosted by the STAS, or are on the ActionList

Use API Post: /V1/Sensors/Lists

With Message Body containing the following fields:

### 6.3.1 Message Body for API Post

| Field | Type | Description | M |
|---|---|---|---|
| ServiceId | Number | Service ID provided by Local EcoSpace | Y |
| SensorId | String | Sensor ID provided by Local EcoSpace | Y |

### 6.3.2 Expected response

| Field | Type | Description |
|---|---|---|
| List | Object: | Collection of ListItems. See 6.3.2.1 |
| Signature | String | Hub generated signature over the response. See 6.5 |

The list is binary ordered on the TokenHash

#### *6.3.2.1 ListItem*

| Field | Type | Description |
|---|---|---|
| TokenHash | String | Base64 encoded hash |

| | | |
|---|---|---|
| TokenType | String | Token Type. "GST" for GST |
| ListType | String | W for whitelist, B for Blacklist, empty for no value. |
| ActionList | Object: | Collection of ActionListItems. See 6.3.2.2 |

#### *6.3.2.2 ActionlistItem for GST*

| Field | Type | Description |
|---|---|---|
| ActionType | String | Type of action. "APDU" for GST |
| APDUValue | String | Base64 encoded APDU |

## 6.4 Send Trigger Message

### 6.4.1 Trigger Pair Token ID / Terminal

After a transaction with a GST and optionally after offline risk management, the STAS sends a Trigger Message to the Hub.

The Trigger Message is constructed by the STAS, using the general construct of the Trigger Message and any extra items that may be required by the service (these are specified in the service's property bag) the Trigger Message is attempting to provision.

Once a trigger message is received, the Hub then routes it to the correct service or services for further processing.

Use API Post: /V1/Trigger

With Message Body containing the following fields:

### 6.4.2 Message Body for API Post

| Field | Type | Description | M |
|---|---|---|---|
| Transaction | Object | Data object See: 6.4.2.1 | Y |
| Tokens | Array | Data Object see: 6.4.2.2 | Y |
| Sensor | Object | Data Object see: 6.4.2.4 | Y |
| Service | Object | Data Object see: 6.4.2.7 | Y |
| ServiceRequestData | Object | Data Object see: 6.4.2.8 | Y |

#### *6.4.2.1 Data Object: Transaction*

| Field | Type | Description | M |
|---|---|---|---|
| TransactionId | String | TransactionId based on localtimestamp, format: yyyyMMddHHmmssfff | Y |

| | | | |
|---|---|---|---|
| Counter | Number | Counter of the sensor, incremented by one per transaction attempt | Y |
| SensorId | String | Sensor ID provided by Local EcoSpace | Y |
| ReferencedTransaction | String | Reference to previous online trigger message (for rollback) | N |
| ExternalTransactionId | String | Sensor reference number, Max Length 40 CHAR | N |

#### *6.4.2.2 Data Object: BaseToken*

| Field | Type | Description | M |
|---|---|---|---|
| TokenType | String | Type of token. Here "GST" | Y |
| TokenValue | String | Unique to abovementioned Token Type e.g. serial number, Token ID. Max Length 255 | Y |
| Propertybag | Object | Data Object see: 6.4.2.3<br>Used for additional Data related to<br>Token e.g. TMAC | N |

#### *6.4.2.3 Data Object: Propertybag*

| Field | Type | Description | M |
|---|---|---|---|
| Key | String | Name of the key e.g. TMAC. Engraved ID, Barcode, TSI | Y |
| Value | | Basic types are supported like, numbers, strings, arrays, objects etc. | Y |

#### *6.4.2.4 Data Object: BaseSensor*

| Field | Type | Description | M |
|---|---|---|---|
| Identifiers | Array | Data Object see: 6.4.2.5 | Y |
| SensorLocation | Object | Data Object see: 6.4.2.6 | N |

#### *6.4.2.5 Data Object: SensorIdentifier*

| Field | Type | Description | M |
|---|---|---|---|
| IdentifierType | String | example.: IMEI / SNR / MAC / etc. Can be nullable.<br>Max Length: 50 | Y |

| | | | |
|---|---|---|---|
| IdentifierValue | String | e.g. serial number, ID. Can be nullable<br>Max Length 255 | Y |

#### 6.4.2.6 Data Object Geolocation

| Field | Type | Description | M |
|---|---|---|---|
| Latitude | String | DD.dddddd° notation | N |
| Longitude | String | DD.dddddd° notation | N |
| Altitude | Number | plus or minus sea level (in meters) | N |
| CellId | Number | A GSM Cell ID (CID) is a generally unique number used to identify each Base transceiver station (BTS) or sector of a BTS within a Location area code (LAC) if not within a GSM network. | N |
| LocationAreaCode | Number | Location area code (LAC) which is a 16 bit number thereby allowing 65536 location areas within one GSM PLMN. | N |
| MobileCountryCode | Number | The Mobile Country Code (MCC) is a three-digit number that used in combination with a Mobile Network Code (MNC) to identify a mobile network operator uniquely. | N |
| MobileNetworkCode | Number | The Mobile Network Code (MNC) is a two digit code (North America) or three digit code (European Standard) that is used in combination with a Mobile Country Code (MCC) to identify a mobile network operator uniquely. | N |

#### 6.4.2.7 Data Object: BaseService

| Field | Type | Description | M |
|---|---|---|---|
| ServiceId | Number | Service ID provided by Local EcoSpace | Y |

#### 6.4.2.8 Data Object: ServiceRequestData

| Field | Type | Description | M |
|---|---|---|---|
| RequestExternalIpAddress | String | External IP-Address of sensor device<br>Example IPv4: 2.2.2.2 | N |
| RequestInternalIpAddress | String | Internal IP-Address of sensor device | N |

| Field | Type | Description | M |
|---|---|---|---|
| | | Example IPv4: 2.2.2.2 | |
| RequestSensorLocalTimestamp | String | Local timestamp | Y |
| Amount | Number | In cents, default 0 | Y |
| CurrencyCode | String | ISO 4217 standard,<br>3 CHAR | Y |
| RequestMode | Number | 1 = Online<br>2 = StoreAndForward | Y |
| AutonomousResult | Number | 0 = ok<br>2 = signature verification failed<br>3 = blacklisted<br>4 = Token expired<br>5 = Token issuer not supported<br>6 = Token status denied | N |
| PropertyBag | Object | Data Object see: 6.4.2.3 | N |

#### *6.4.2.9 Return messagetype: ResponseMessage*

| Field | Type | Description |
|---|---|---|
| Transaction | Object | Identical structure and values to the request. |
| Message | String | Optional message |
| ResponseValue | Number | For values see: 6.4.3 |
| PropertyBag | Object | Data Object see: 6.4.2.3 |
| Signature | String | Hub generated signature over the response. See 6.5 |

### 6.4.3 Response Messages

| Message | ResponseValue | Description |
|---|---|---|
| SUCCESS | 0 | TriggerRequest was successfully handled by EcoHub. |
| UNKNOWN SENSOR | -2 | Sensor used to send triggermessage is not an active Sensor in the EcoHub. |
| TRANSACTIONRECEIPT CHECK FAILED | -3 | Validation of TMAC of triggerrequest failed. |

| Message | ResponseValue | Description |
|---|---|---|
| UNKNOWN TOKEN FOR SERVICE | -4 | There is no subscription for the requested service. |
| UNKNOWN SERVICE | -5 | Requested service is not registered |
| REQUESTMODE <x> NOT SUPPORTED FOR REQUESTED SERVICE | -6 | The requestmode (online or storeAndForward) is not supported for requested service |
| TOKENTYPE NOT ALLOWED FOR SERVICE | -7 | Requested service does not support tokentype |
| TOKEN IS NOT REGISTERED | -8 | Unknown token used. Tokens need to be registered at Hub before they can be used. |
| NO SERVICE-ENDPOINT FOUND | -9 | No endpoint available in cache for generic servicehandler |
| <General Error> | -1 | Message contains the error. |

## 6.5 Verification of Hub response messages

### 6.5.1 Signature verification

All response messages from the Hub are signed by the Hub. The data signed comprises the entire response message except the signature value itself.

The following section is under construction:

The signature itself consists a Base64 encoding of ASN.1 DER structured r and s fields of the signature:

```
ECDSASignature ::= SEQUENCE {
    r   INTEGER,
    s   INTEGER
}
```

The ECDSA signature SHALL be verified according to [X9.62, §7.4], using SHAxxx as the hash function and elliptic curve domain parameters as specified by [RFC 5639, § 3.3] with fixed Curve-ID brainpoolPxxxxx. The public key to be used is the Hub public key.

If the signature consists of zero bytes (0x00), the STAS shall decline the transaction.

## 6.6 Error codes

If a Sensor API call fails, the API returns an error code based on the standardized HTTP error codes

See https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Used error codes:

| Error code | Description / Example |
|---|---|
| 400 | Bad request – Invalid model (data send to the API does not meet minimal requirements) |
| 400 | Bad request - serviceId needs to be a positive number. |
| 400 | Bad request – invalid sensor (sensorid is unknown) |
| 400 | Bad request – no certificate available (signature is set, but certificate is not known for this sensorid) |
| 400 | Bad request – no signature available (no signature is set, but is required by sensorconfiguration) |
| 400 | Bad request – invalid signature (signature does not match the data provided) |
| 500 | Internal server error – general error. |

# Revisions

| Who | Date | Modifications | Version |
|---|---|---|---|
| IB | 31-10-2015 | Initial version | 0.1 |
| GRB | 2-11-2015 | Review | 0.2 |
| IB | 3-11-2015 | Process review | 0.3 |
| IB | 30-11-2015 | Major changes to the document structure | 0.4 |
| IB | | Many updates | 0.95 |
| IB | 11-12-2015 | Review corrections | 1.0 |

# Appendix A. Certificate profiles

## Body

| Certificate Component | Reference [RFC5280] | Presence | comments |
|---|---|---|---|
| Certificate | 4.1.1.1 | M | |
| TBSCertificate | 4.1.2 | M | |
| Version | 4.1.2.1 | M | v3 ('02'h) |
| serialNumber | 4.1.2.2 | M | Certificate Serial Number |
| signature | 4.1.2.3 | M | See Algorithms |
| issuer | 4.1.2.4 | M | See Naming conventions |
| validity | 4.1.2.5 | M | - |
| subject | 4.1.2.6 | M | See Naming conventions |
| subjectPublicKeyInfo | 4.1.2.7 | M | See Algorithms |
| issuerUniqueID | 4.1.2.8 | x | |
| subjectUniqueID | 4.1.2.9 | x | |
| extensions | 4.2 | M | See Extensions |
| signatureAlgorithm | 4.1.1.2 | M | See Algorithms |
| signatureValue | 4.1.1.3 | M | |

*Table 9 Main structure of the certificate*

## Naming conventions

The subject and issuer names are encoded in UTF8 and follow the naming conventions as described in the table below. Note that for Token certificates the common name shall be equal to the hexadecimal representation of the TokenID prepended with "0x", for example: cn=0x00102030405060708090.

| Entity | Organization Name | Organizational Unit Name | Common Name | Serial Number |
|---|---|---|---|---|
| Root CA | | | | |
| Sub-CA | o=European Travelers Club | ou=D/T/A/P* | cn=<Sub-CA ID> | serialNumber=<Cert Serial Number> |
| Token | o=European Travelers Club | ou=D/T/A/P* | cn=<TokenID> | serialNumber=<Cert Serial Number> |
| | | | | |

*Table 10 Naming conventions of the subject and issuer names*

* D = Development environment, T=Test environment, A=Acceptance environment, P=Production environment

## Extensions

The table below indicates which extensions are present (M) and critical (C) in the different types of certificates.

| Extension Name | Reference [RFC5280] | Sub-CA | Token |
|---|---|---|---|
| AuthorityKeyIdentifier | 4.2.1.1 | M | M |
| keyIdentifier | | M | M |
| authorityCertIssuer | | x | x |
| authorityCertSerialNumber | | x | x |
| SubjectKeyIdentifier | 4.2.1.2 | M | x |
| subjectKeyIdentifier | | M | x |
| KeyUsage | 4.2.1.3 | M | M |
| digitalSignature | | x | M |
| nonRepudiation | | x | x |
| keyEncipherment | | x | x |
| dataEncipherment | | x | x |
| keyAgreement | | x | x |
| keyCertSign | | M | x |
| cRLSign | | M | x |
| encipherOnly | | x | x |
| decipherOnly | | x | x |
| CertificatePolicies | 4.2.1.4 | x | x |
| PolicyMappings | 4.2.1.5 | x | x |
| SubjectAltName | 4.2.1.6 | x | x |
| IssuerAltName | 4.2.1.7 | x | x |
| SubjectDirectoryAttributes | 4.2.1.8 | x | x |
| BasicConstraints | 4.2.1.9 | MC | M |
| cA | | 'FF' True | x, False |
| PathLenConstraint | | 0 | x |
| NameConstraints | 4.2.1.10 | x | x |

| Extension Name | Reference [RFC5280] | Sub-CA | Token |
|---|---|---|---|
| PolicyConstraints | 4.2.1.11 | x | x |
| ExtKeyUsage | 4.2.1.12 | x | x |
| CRLDistributionPoints | 4.2.1.13 | M | x |
| distributionPoint | | M | x |
| reasons | | x | x |
| cRLIssuer | | x | x |
| InhibitAnyPolicy | 4.2.1.14 | x | x |
| FreshestCRL | 4.2.1.15 | x | x |
| privateInternetExtensions | 4.2.2 | x | x |

*Table 11 Extensions used in the different certificate types*

## Algorithms

| Entity | ECDSA key size | Hashing Algorithm for Certificate Signing |
|---|---|---|
| Root CA | 256 | SHA256 |
| Sub-CA | 256 | SHA256 |
| Token | 224 | SHA224 |

*Table 12 Algorithms used for signing per certificate type*