For ESDC developers

E-SDCs are software applications or hardware devices whose main function is to:

- fiscalize invoices,
- enable issuing invoices with or without the internet (by safeguarding invoices in its memory)
- deliver fiscalized audit packages to the tax authority
- enable the audit of fiscalized invoices

From the technical point of view, E-SDC is a middleware component that connects an accredited invoicing system (POS) to a secure element and enables standardized communication with TaxCore.API.

This article offers useful initial information about technical requirements for accrediting E-SDC solutions.

NOTE:
Before reading the rest of this article, make sure you have read Getting Started With Accreditation
Also, before you start developing your solution, please read General Information for all vendors.

This article offers E-SDC vendors who are interested in accrediting an E-SDC solution the following insight:

- quick guide for E-SDC accreditation
- desired/needed professional experience for developing an E-SDC product
- how an E-SDC solution fits in with other EFD components?
- how to navigate the rest of the technical instructions in order to initialize development?

Quick step-by-step guide for E-SDC accreditation

1. Register as a vendor for the Sandbox environment
2. Receive a Developer Certificate and use it to access the Developer Portal
3. Use the Developer Portal to request additional certificates (smart cards) for testing purposes
4. Consult all the sections in these technical instructions to see understand all the requirements and how they should be implemented
5. Use the testing application SDC Analyzer on the Developer Portal to test your E-SDC's operation
6. Compile user documentation for your POS
7. Use the My Accreditations section on the Developer Portal to apply for accrediting your POS

Which professional experience do I need for developing an E-SDC?

Be mindful that developing E-SDCs is much more challenging than developing POS solutions.

For that reason, we have compiled a list of desired professional experiences in order to prepare developers for the challenges that an E-SDC development puts forwards:

- a clear understanding of the multi-environment development concept - E-SDCs are developed in the

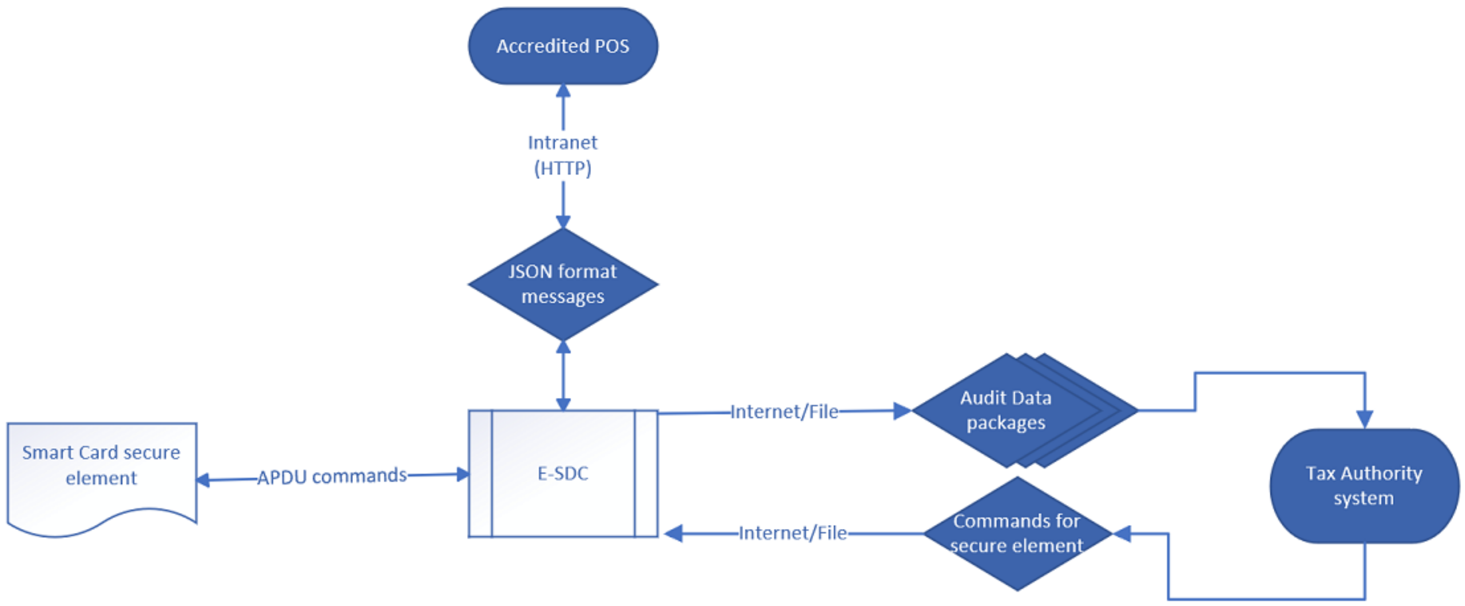
development Sandbox environment and used by taxpayers in the official Production environment
- using APDU commands for communication with secure elements
- using Client Certificate (certificate store, pkcs12, pkcs11) authentication via HTTP protocol
- calling Web API
- using RSA and AES algorithms for encryption
- parsing X.509 certificates and using the obtained data

Integration with other EFD components

E-SDC is one of three components of any EFD setup.

NOTE:
For an overview of all EFD components and how they communicate with each other, see Electronic Fiscal Device.
For an overview of how EFD components interact to create a fiscal invoice, see Fiscal Invoice.

The graph below shows the communication pattern that an E-SDC establishes with other EFD components.



What is the typical process flow of E-SDC operations?

All E-SDC solutions must follow the basic steps in the process of configuration and creating fiscal invoices (although some might have additional, manufacturer-specific, steps).

For a detailed step-by-step explanation of all standard E-SDC operations, see Processes.

NOTE:
Be mindful that E-SDC must be initialized every time when a new smart card is used. For more details, see E-SDC Initialization.

Communication with an accredited invoicing system (POS)

E-SDC communicates with an accredited invoicing system (POS) to receive invoice requests and return invoice responses - as part of fiscalizing each invoice.

Communication between a POS solution and E-SDC is established through JSON formatted messages using the POS to SDC Protocol.

Communication with a secure element

E-SDC communicates with a secure element (issued to taxpayers a smart card) to:

- perform fiscalization of data submitted as an invoice request from POS
- enable the audit of invoices fiscalized by that secure element
- enable transmission of other messages exchanged between the secure element and the tax authority (e.g. transferring new tax rates to the secure element)

Communication between an E-SDC and a secure element is established using APDU Commands.

For authentication purposes, an E-SDC also must be able to communicate with the PKI Applet installed on each smart card secure element. For more information, see PKI Applet.

Communication with the TaxCore.API (tax authority system)

E-SDC communicates with TaxCore.API to:

- submit fiscalized audit packages to the TaxCore database
- receive commands from TaxCore (which it then forwards to the secure element)

Communication between E-SDC and tax authority is established via TaxCore.API.

File-based communication

In case of prologued internet failure, E-SDC must allow file-based communication with the tax authority's system for the purpose of configuration and performing local audits.

All sections of the technical instructions

Technical instructions specific for E-SDC vendors/developers consist of the following sections:

1. High Level Requirements
   This section describes high-level requirements to consider when

2. Architecture
   This figure shows the components of a fiscalization system and their mutual relationship.

3. Data Structures
   This section contains descriptions of the main data structures used during the fiscalization and audit processes.

4. Processes
   This section describes processes performed by an E-SDC.

5. Protocols
   This section describes Application Programming Interfaces (API) and protocols exposed by an E-SDC or used by an E-SDC to communicate with the other components (TaxCore.API, Secure element Applet, PKI Applet or SD Card/USB Flash Drive) required to fulfill its primary role – to safeguard a transaction and to transfer the audit packages to the tax authority’s system.

6. Manuals
   E-SDC must have a user manual that explains the following topics in detail:

Related articles

- EFD Vendors General Information

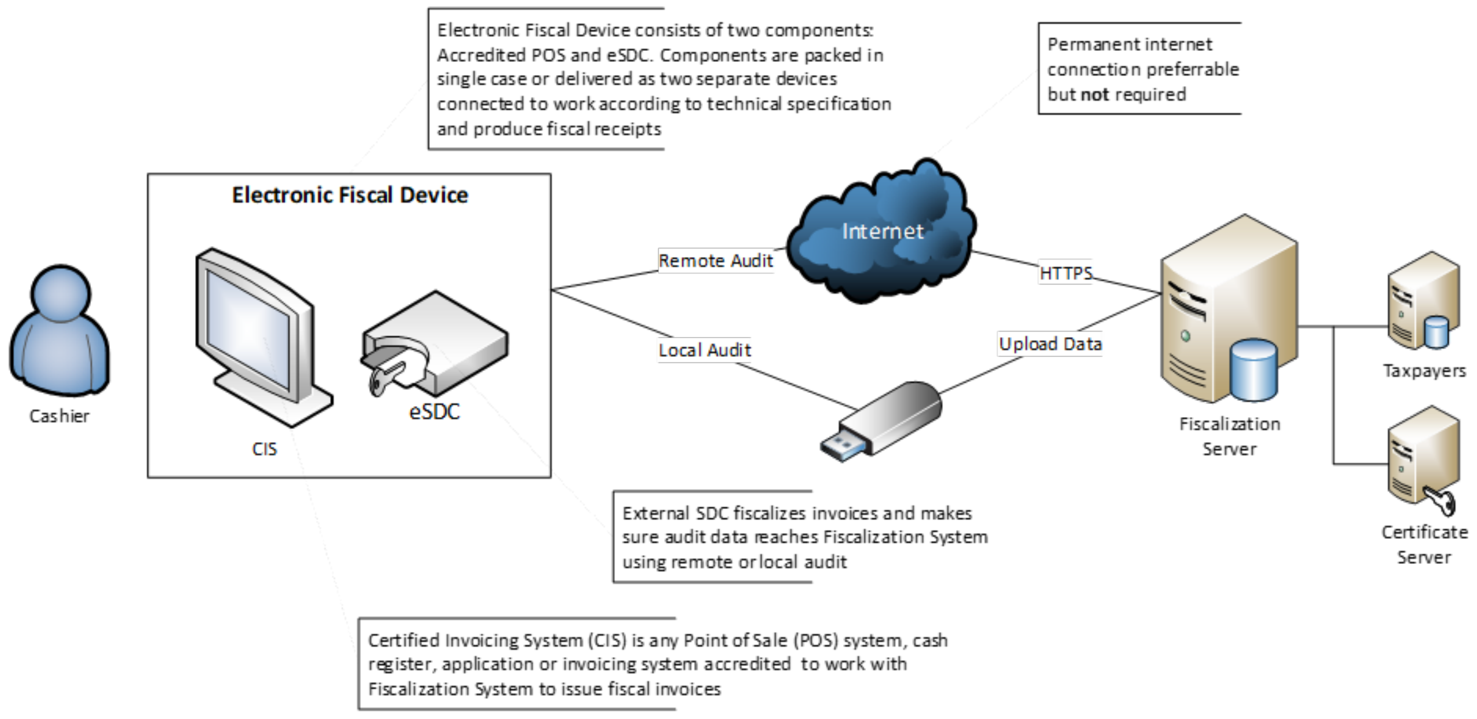
High-Level Requirements

This section describes high-level requirements to consider when designing an E-SDC.

1. E-SDC is able to generate an invoice without an Internet connection.
2. E-SDC relies on a Secure Element (delivered as a smart card) to safeguard an invoice.
3. E-SDC must be designed so that it can work in multiple environments (development and production), depending on the certificate on the secure element it is using.
4. E-SDC calculates tax liabilities based on received Invoice items and defined tax rates.
5. Initial E-SDC configuration can be performed via a file (e.g. on USB disk) or the Internet connection.
6. E-SDC exposes a standardized protocol to a POS - JSON via HTTP.
7. E-SDC processes all commands received from the tax authority's system in consecutive order. These commands can include time synchronization, an update of tax rates, etc.
8. E-SDC encrypts audit data and stores it locally in an encrypted form.

9. E-SDC periodically sends stored audit data to a tax authority and this process is called an audit.
10. E-SDC does not have to keep audit data submitted and successfully stored on the tax authority's system.
11. E-SDC submits a proof of audit (PoA) generated by the Tax Service's system to the Secure Element as soon as the E-SDC receives it.
12. E-SDC does not store the Secure Element's PIN code except in the working memory. Once the E-SDC is restarted, the cashier is required to enter the PIN code again.
13. E-SDC stores Audit Packages in its non-volatile memory for the length of the period prescribed by the governing tax authority.
14. E-SDC keeps a log of all required error events with the exact local date and time

This figure shows the high-level architecture of Fiscalization System, involving an Accredited POS, E-SDC and tax authority's system (TaxCore).


Electronic Fiscal Device consists of two components: Accredited POS and eSDC. Components are packed in single case or delivered as two separate devices connected to work according to technical specification and produce fiscal receipts
Permanent internet connection preferrable but **not** required
Electronic Fiscal Device
Internet
Remote Audit
HTTPS
Local Audit
Upload Data
Cashier
CIS
eSDC
Fiscalization Server
Taxpayers
Certificate Server
External SDC fiscalizes invoices and makes sure audit data reaches Fiscalization System using remote or local audit
Certified Invoicing System (CIS) is any Point of Sale (POS) system, cash register, application or invoicing system accredited to work with Fiscalization System to issue fiscal invoices

Connectivity And Modes of Operation

Introduction

External Sales Data Controller (E-SDC) device exposes HTTP protocol for communication with an Accredited POS via a UTP cable, Wi-Fi and similar.

E-SDC uses a Secure Element to digitally sign invoices received from the Accredited POS and to produce audit data. The audit data is stored on the E-SDC’s internal non-volatile memory, which enables a local and remote audit.

Multiple POSs can be connected to a single E-SDC. However, this should be avoided as multiple devices could send the data simultaneously, and since every smart card has its own limitations (resources, processing speed), that could slow down the overall process at sale point.

Use an online mode whenever possible

Taxpayers are encouraged to use an online mode whenever it's possible - V-SDC service will be widely available and accessible for a variety of Accredited POS devices and software solutions. But, in order to rollout, a fiscalization system needs to have the ability to close any possible gap in the fiscal discipline due to poor network coverage or the Internet unavailability.

Semi-Connected mode

E-SDC must be able to work in the Semi-offline mode.

In the semi-offline mode, the E-SDC can work both online (if the Internet is available) and offline (if the Internet is not available).

In the semi-offline mode, the Secure Element signs an invoice and the E-SDC device immediately tries to contact the tax authority's system and perform a remote audit by invoking Submit Audit Package web method.

If the tax authority's system is not accessible (for example, the internet connection is interrupted), the E-SDC automatically switches to the offline operation. In the offline operation, the Secure Element signs an invoice and the E-SDC device stores an audit package locally in a secure manner.

When the connection with the tax authority system is re-established, the E-SDC will send the locally stored audit packages.

If the connection is interrupted for a prolonged period, the audit packages will be retrieved through a local audit process.

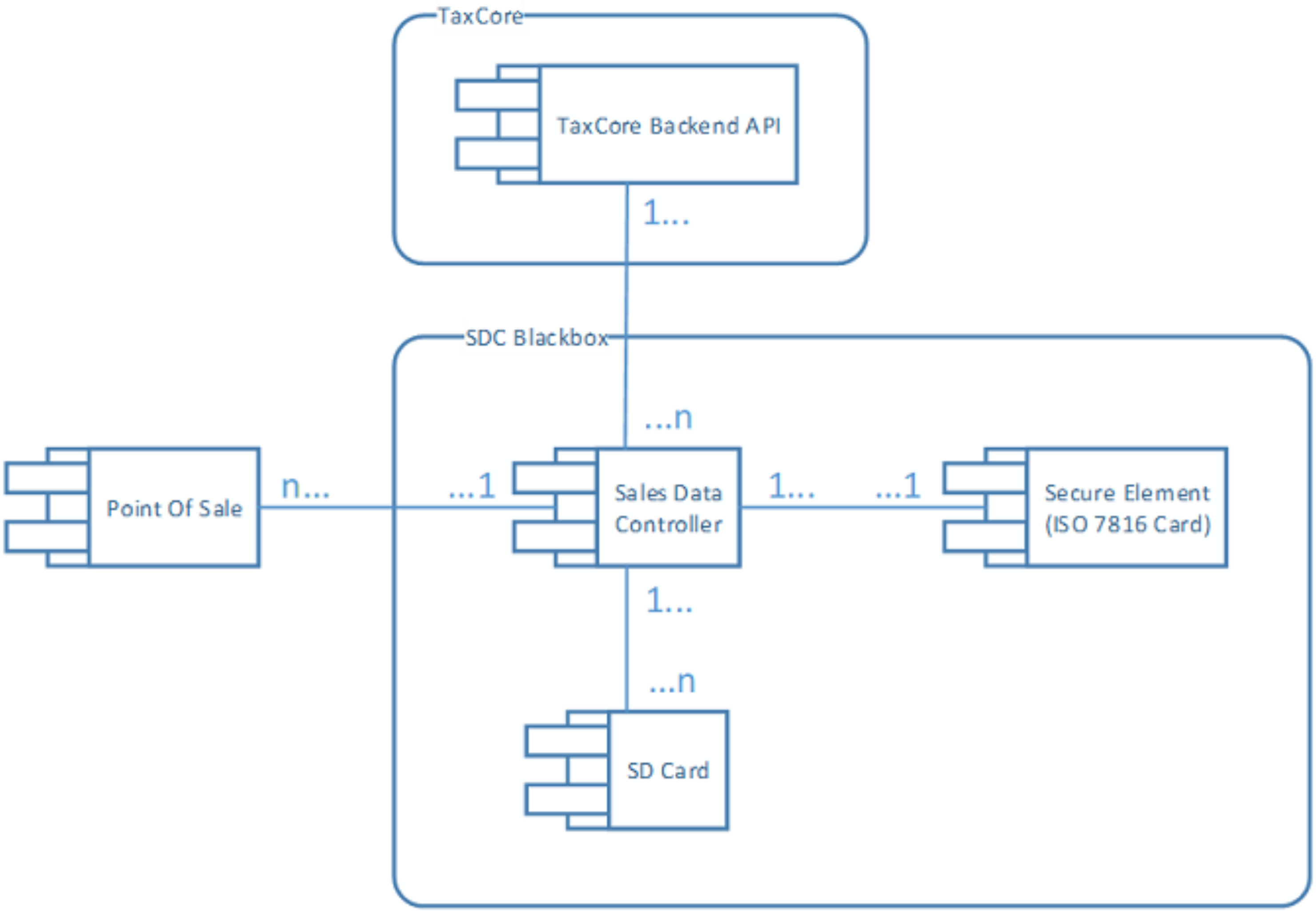
Architecture

Content

Smart Cards
For standard operations, each E-SDC requires a Smart card issued by a tax authority, which consists of two applets:

3.
E SDC States
The diagram below shows the basic E-SDC states and transitions.

This figure shows the components of a fiscalization system and their mutual relationship.



ESDC Architecture – Image showing the components of a fiscalization system and their mutual relationship

E-SDC Implementation

E-SDC can be implemented as a hardware device or a software service depending on the E-SDC manufacturer's decision and clients' infrastructure. E-SDC can also be implemented as an integral part of a POS.

In any of those cases, the E-SDC component has to pass the same accreditation process and prove that E-SDC is implemented according to the instructions described in this document.

1.

Ports
Each E-SDC must provide ports or connectivity for the following purposes

2. Standards
E-SDC device complies with all current local regulations regarding safety usage, electromagnetic compatibility, temperature range and power supply.

3. Power Supply
It is allowed to use both AC and DC voltage for power supply. In the case of the AC voltage, a device shall work with the frequency range 50-60 Hz. The power supply circuit used by the E-SDC shall be protected with an automatic circuit breaker, suitable for electronic devices (type I).

Ports

Each E-SDC must provide ports or connectivity for the following purposes

Secure Element

E-SDC has a smart card reader in compliance with ISO/IEC 7810 and ISO/IEC 7816 standard. Smart Card reader may be internal to the E-SDC device or external usually connected via USB port with computer running E-SDC software.

The supported Smart Card sizes are 1FF (credit card size) and 2FF (mini SIM card size).

SIM card	Standard reference	Length (mm)	Width (mm)	Thickness (mm)	Volume (mm^3)
Full-size (1FF)	ISO/IEC 7810:2003, ID-1	85.60	53.98	0.76	3511.72
Mini-SIM (2FF)	ISO/IEC 7810:2003, ID-000	25.00	15.00	0.76	285.00

Audit

If the E-SDC device uses a USB flash drive for a Local Audit, USB connectors "USB Type B female" must be used.

Applied USB communication protocol shall be “USB 2.0” or higher.

For situations when an SD Flash memory card is used for a Local audit, a device must have an easily accessible Micro SD card connector.

For a Remote Audit, these Instructions do not limit a manufacturer in choosing a communication port as long as the invoice signing is not interrupted.

POS Connectivity

POS must be able to connect to the E-SDC using at least one of the following ports:

Ethernet

Ethernet port in compliance with IEEE 802.3 standard, present on an E-SDC device. The minimum speed of the Ethernet port is at least 10 Mb/s.

Wireless

Wireless connection in compliance with IEEE 802.11 (Wi-Fi/Bluetooth) to a POS device and a local network.

Standards

E-SDC device complies with all current local regulations regarding safety usage, electromagnetic compatibility, temperature range and power supply.

Particularly, in terms of electromagnetic compatibility, a product shall comply with the following standards

- EN 55022:2010
- EN 55032:2012
- EN 55024:2010 A1:2015.

A device's operational temperature range shall be 0° - 70° C (Commercial range).

Power Supply

It is allowed to use both AC and DC voltage for power supply. In the case of the AC voltage, a device shall work with the frequency range 50-60 Hz. The power supply circuit used by the E-SDC shall be protected with an automatic circuit breaker, suitable for electronic devices (type I).

In order to protect sensitive electronic components, the smart card and data stored on non-volatile memory, besides the mandatory basic protection, it is recommended that device is equipped with the additional fast overcurrent protection in the form of a fast fuse, e-fuse or similar device with the short time of operation.

When the DC voltage is used, protection against the reverse polarity shall be applied.

If the power supply voltages are higher than 75 Vdc or 50 Vac, a manufacturer shall obtain the appropriate certificate from a local authority, or represent a certificate valid in the country of use.

Smart Cards

For standard operations, each E-SDC requires a Smart card issued by a tax authority, which consists of two applets:

- Secure element Applet - used to apply a digital signature and maintain a set of fiscal counters in the offline mode
- PKI Applet - used to authenticate and establish a secure connection with the TaxCore.API web service

Both applets share the same PIN code. PIN is chosen during the process of requesting an Admin Secure element (smart card) - see Registration for Developer Portal- or while requesting an additional POS secure elements - see Requesting Additional Certificates.

PIN can't be changed after it is selected.

Each smart card is uniquely identified by a UID - Unique Identifier. Each digital certificate issued for E-SDCs has UID embedded in the certificate’s subject field.

1. Secure Element Applet
   Secure element (SE) is a fiscal component implemented as a special software or a device designed to receive invoice data, perform signing and data processing and generate a response which is sent back to the caller for further actions.

2. PKI Applet
   PKI Applet contains a digital certificate and a private key used to authenticate E-SDC to TaxCore.API web services.

Secure Element Applet

Introduction

Secure element (SE) is a fiscal component implemented as a special software or a device designed to receive invoice data, perform signing and data processing and generate a response which is sent back to the caller for further actions.

This response provides the authenticity of invoice data.

Secure Element is issued and controlled by the tax authority. The main purpose of the SE is to sign invoices using a taxpayer’s digital certificate, control audits and maintain a set of fiscal counters.

Each taxpayer is uniquely identified using digital certificates based on the Public Key Infrastructure (PKI).

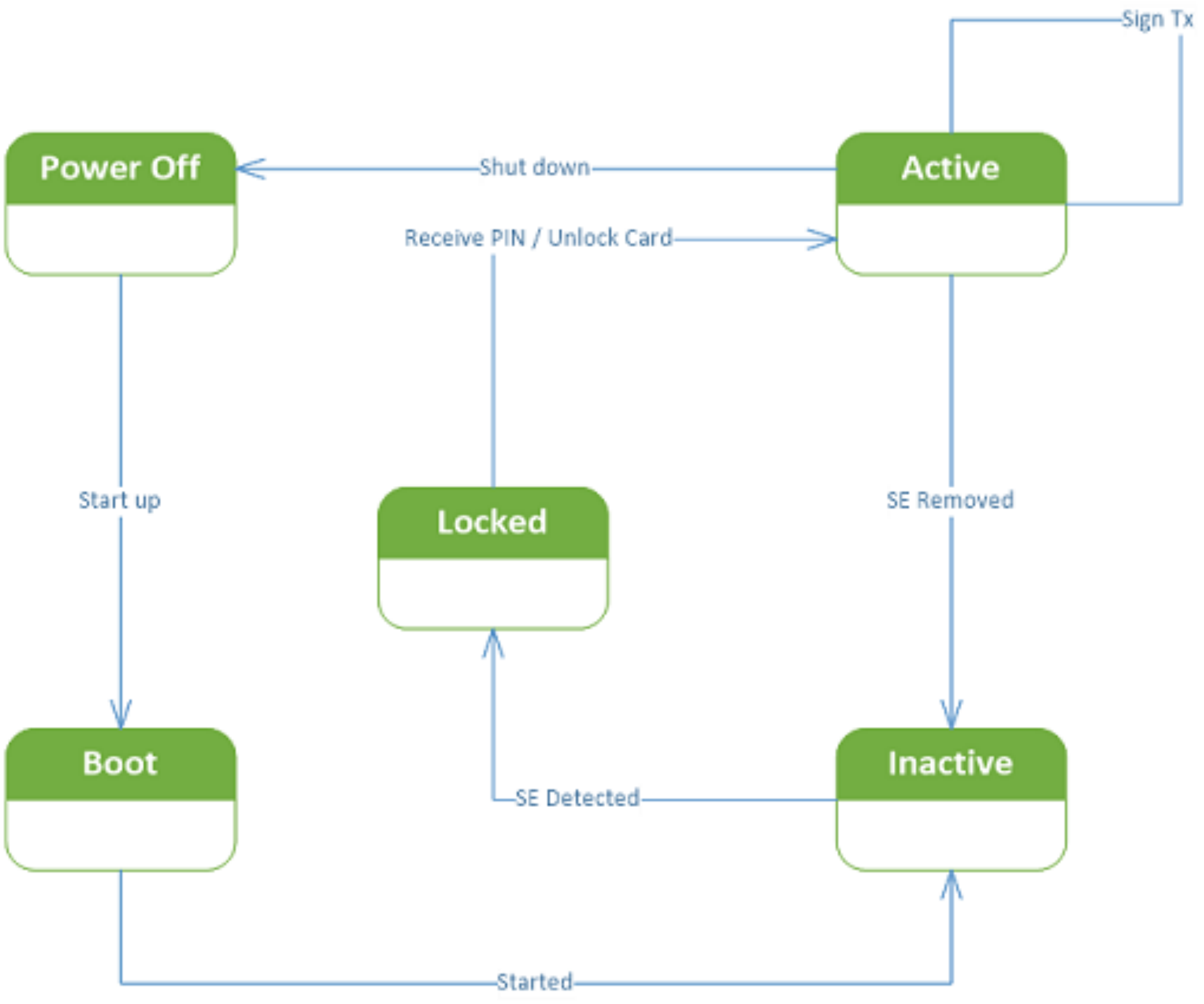
Secure element will stop issuing invoices if the maximum allowed amount for that particular fiscal device is exceeded – this facilitates the regular audit and forces taxpayers to report back to the tax authority system. Likewise, the SE will continue to produce fiscal invoices once it receives proof from TaxCore.API that audit has been received and stored on the tax authority’s system.

PKI Applet

PKI Applet contains a digital certificate and a private key used to authenticate E-SDC to TaxCore.API web services.

E-SDC States

The diagram below shows the basic E-SDC states and transitions.



E-SDC States – Image showing the basic E-SDC states and transitions

Data Structures

This section contains descriptions of the main data structures used during the fiscalization and audit processes.

1. Date and Time
   E-SDC has access to the current time. Real-Time Clock or a similar component must be installed and used to maintain the correct time while the power is off. In case of a power outage, the Real-Time Clock must be able to operate without interruption for up to 6 months.

2. Audits
   Audit data represent a machine-readable formatted fiscal invoice signed by a taxpayer's private key followed by journal data. Journal data is a textual representation of a fiscal invoice generated by E-SDC.

3.

Commands
Commands are a means of communication between the tax authority's system and E-SDCs. Commands are stacked in the queue list on the server for a specific E-SDC and submitted to the E-SDC as a part of the response once it reports to the tax authority's system using a Remote or a Local audit.

Date and Time

E-SDC has access to the current time. Real-Time Clock or a similar component must be installed and used to maintain the correct time while the power is off. In case of a power outage, the Real-Time Clock must be able to operate without interruption for up to 6 months.

E-SDC synchronizes the time with the NTP Server, configured by the *Configure Time Server URL Command*.

UTC standard is used only in the `sdcDateTime` value when creating an audit package to be sent to TaxCore.API.

Local time is used by E-SDC for all other purposes, such as:

- Date and time sent by a POS to an E-SDC
- Date and time printed on a journal (textual representation of an invoice) generated by an E-SDC
- Date and time sent by an E-SDC to a POS

JSON-based protocols use date and time according to ISO 8601 where applicable. For example:

Date: **2019-12-11**

Date and time in UTC:

2019-12-11T10:06:33+00:00

2019-12-11T10:06:33Z

20191211T100633Z

Week:

2019-W50

Date with week number:

2019-W50-3

Date without year:

--12-11[1]

Ordinal date:

2019-345

Time offsets are represented in format “**LocalTime+/-Zone**” as in case of **2019-12-11T10:06:33+00:00**

NOTE:
Date and Time display format on all TaxCore portals is: "dd/MM/yyyy HH:mm:ss" (e.g. 15/10/2018 14:28:24).

All binary-based protocols E-SDC to Secure element - APDU commands use Unix Timestamp format formatted as a 64bit unsigned integer Big Endian (for example: 1495018011910 is 2017-05-17T10:46:51.910Z).

NOTE:
Be mindful of the minimum (1900-01-01T00:00:01Z) and maximum (9999-12-31T23:59:59Z) values.
Also, individual tax jurisdictions might have specific requirements regarding the maximum difference between the time of invoice creation (SDC time) and the moment when an audit package arrives to СУФ Развој. Make sure you are familiar with those requirements.

Audits

Audit data represent a machine-readable formatted fiscal invoice signed by a taxpayer’s private key followed by journal data. Journal data is a textual representation of a fiscal invoice generated by E-SDC.

Content of audit data is kept in encrypted form (audit package) ensuring no changes have been made and that no one has been able to access its content after creation, except the tax authority’s system, after a successful audit process.

1. Format of Audit Data
   Invoice created as described in section Create Invoice represents audit data. Audit data is a JSON representation of an invoice that consists of two separate data structures:

2. Format of the Audit Proof Request
   The audit-proof request is a textual file in JSON format:

3. Format of the Audit Package
   A file containing encrypted audit data is called an Audit Package. Encryption of audit data prevents access to sales data by unauthorized persons.

Format of Audit Data

Invoice created as described in section Create Invoice represents audit data. Audit data is a JSON representation of an invoice that consists of two separate data structures:

`InvoiceRequest` – this object is created by POS and submitted to E-SDC

`InvoiceResponse` – this object is created by E-SDC and returned to POS

Model

This Model is designed and based on OpenAPI-Specification V2 (https://github.com/OAI/OpenAPI-Specification).

```
Invoice {
    request (InvoiceRequest),
    result (InvoiceResult)
}
```

Example

NOTE:
All field values in the example are for informative purposes only.

```
{
□"request": {
□□"dateAndTimeOfIssue": "2021-09-02T14:45:59",
□□"cashier": "Software QA Engineer",
□□"buyerId": "DT12345",
□□"buyerCostCenterId": "Data Tech",
□□"invoiceType": 0,
□□"transactionType": 1,
□□"payment": [
□□□{
□□□□"amount": 5.0,
□□□□"paymentType": 2
□□□},
□□□{
□□□□"amount": 20.0,
□□□□"paymentType": 1
□□□}
□□],
□□"invoiceNumber": "07-2021-3",
□□"referentDocumentNumber": "BQVWAAR4-BQVWAAR4-23",
□□"referentDocumentDT": "2021-09-02T13:45:29",
□□"options": {
□□□"omitTextualRepresentation": "0",
□□□"omitQRCodeGen": "1"
□□},
□□"items": [
□□□{
□□□□"gtin": "987654321",
□□□□"name": "Mate 20 Lite",
□□□□"quantity": 2.0,
□□□□"discount": null,
□□□□"labels": [
□□□□□"A",
□□□□□"B"
□□□□],
□□□□"unitPrice": 10.0,
□□□□"totalAmount": 20.0
□□□},
□□□{
□□□□"gtin": "987654321",
□□□□"name": "Credit",
□□□□"quantity": 1.0,
□□□□"discount": null,
□□□□"labels": [
□□□□□"A"
□□□□],
□□□□"unitPrice": 5.0,
□□□□"totalAmount": 5.0
□□□}
□□]
□},
□"result": {
□□"requestedBy": "BQVWAAR4",
□□"sdcDateTime": "2021-09-02T12:46:00.8266132Z",
□□"invoiceCounter": "4/27NR",
□□"invoiceCounterExtension": "NR",
□□"invoiceNumber": "BQVWAAR4-BQVWAAR4-27",
□□"taxItems": [
□□□{
□□□□"categoryType": 0,
□□□□"label": "A",
□□□□"amount": 2.0642,
□□□□"rate": 9.0,
□□□□"categoryName": "VAT"
□□□},
□□□{
□□□□"categoryType": 0,
□□□□"label": "B",
□□□□"amount": 0.0,
□□□□"rate": 0.0,
□□□□"categoryName": "VAT"
□□□}
□□],
□□"verificationUrl": "https://ft8.test.taxcore.dti.rs/v/?vl=A0JRVldBQVI0QlFWV0FBUjQbAAAABAAAA
□□"verificationQRCode": "R0lGODlhlAGUAfcAAAAAAAAAMwAAZgAAmQAAzAAA/wArAAArMwArZgArmQArzAAr/wB\
□□"journal": "============ FISCAL INVOICE ============\r\nTIN:                              RS6
□□"messages": "Success",
□□"signedBy": "BQVWAAR4",
□□"encryptedInternalData": "PtYAAizJaZutdu8gViJuVkh13X2xiYUoSM8APq7Il5/a/Ubtm8Y1aMt+MBG1b5Mpa
□□"signature": "AglDF5QXeQN0sbffN+TiDNVKq3ySAr9KGe2ohOzROfJOHgQ9T68dv+VjA6arNY40gKuLUE8BjqVOZ
□□"totalCounter": 27,
□□"transactionTypeCounter": 4,
□□"totalAmount": 25.0,
□□"taxGroupRevision": 5,
□□"businessName": "Premier League DTI",
□□"tin": "RS654321",
□□"locationName": "Premier League DTI",
□□"address": "Kruzni put 7",
□□"district": "Lestane",
□□"mrc": "00-1002-BQVWAAR4"
□}
}
```

Format of the Audit-Proof Request

The audit-proof request is a textual file in JSON format:

```
ProofOfAuditRequest {
auditRequestPayload (string),
sum (integer) 64bit unsigned,
limit (integer) 64bit unsigned
}
```

Field	Type	Description
auditRequestPayload	Base64 Encoded String	Byte array obtained from the secure element operation Start Audit, encoded as Base64 string
sum	64bit unsigned integer	Sum of SALE and REFUND of Secure Element, as per Amount Status command in Fiscalization
limit	64bit unsigned integer	The Limit Amount of the Secure Element, as per Amount Status command in Fiscalization

Format of the Audit Package

A file containing encrypted audit data is called an Audit Package. Encryption of audit data prevents access to sales data by unauthorized persons.

Only the tax authority's system software running on the premise, and used by authorized personnel, can decrypt audit packages. A one-time symmetric key is newly created for each audit package.

The Audit Package is a textual file in JSON format:

```
AuditPackage {
key (string),
iv (string),
payload (string),
invoiceNumber (string)
}
```

Field	Type	Description
key	Base64 Encoded String	One-time symmetric key (256Bit long) encrypted using RSA with TaxCore public key, obtained from a secure element using APDU command Export TaxCore Public Key
iv	Base64 Encoded String	Initialization vector Key encrypted using RSA and TaxCore

		public key, obtained from a secure element using APDU command Export TaxCore Public Key
payload	Base64 Encoded String	Audit data encrypted with **key** and **iv** using AES256 algorithm (CBC CypherMode and PKCS7 PaddingScheme). It is a Base64Encoded JSON format of an invoice, as described in section Format of the Audit Data.
invoiceNumber	String	SDC Invoice Number of the invoice in format [Requested by UID]-[Signed by UID]-[number]

Commands

Introduction

Commands are a means of communication between the tax authority's system and E-SDCs. Commands are stacked in the queue list on the server for a specific E-SDC and submitted to the E-SDC as a part of the response once it reports to the tax authority's system using a Remote or a Local audit.

Commands are always delivered as an array structure. Commands are executed in consecutive order, starting from the first one in the array.

Below is the data structure of a single command:

```
Command {
      "commandId": (GUID),
      "type": (enum CommandsType),
      "payload": (Json string),
       "uid": (string)
    }

   enum CommandsType
    {
        SetTaxRates = 0,
        SetTimeServerUrl = 1,
        SetVerificationUrl = 2,
        TaxCorePublicKey = 4 // deprecated,
        ForwardProofOfAudit = 5,
        SetTaxCoreConfiguration = 7
        ForwardSecureElementDirective = 8
    }
```

commandId is a unique identifier assigned by tax authority's system. Once a command is successfully executed, TaxCore.API shall be notified as described in the section Notify Command Processed.

type defines the type of command and a format of a Payload as described in the following sections. Valid values are defined by CommandsType enum.

Payload transfers the information form TaxCore.API to an E-SDC in JSON format. The format of each type is described in the following sections.

Commands Execution Results

Commands Execution Results are returned to TaxCore.API after each command is executed. It is a confirmation to TaxCore.API that a certain command is executed. This structure is used only for file-based communication, when E-SDC indirectly (e.g. via TAP) reports commands execution status for multiple commands.

Model

```
[
    {
        "commandId": (GUID),
        "success": (boolean),
        "dateAndTime": (string)
    }
]
```

Example

```
[
    {
        "commandId": "945bb863-5c7f-4826-9ae3-26debcac331a",
        "success": true,
        "dateAndTime": "2017-06-17T04:33:47+00:00"
    }
]
```

Command Types

0 - Set Tax Rates Command

1 - Set Time Server URL Command

2 - Set Verification URL Command

5 - Forward Proof of Audit Command

7 - Set TaxCore Configuration Command

8 - Forward Secure Element Directive

Set Time Server URL Command

E-SDC updates the URL of the time server used to keep a local clock in sync. Payload is the URL of the target NTP server.

NOTE:
This command is an *initialization* command during E-SDC's first communication with the Tax Service system. After that, it is an *update* command for every next communication.

Example

0.europe.pool.ntp.org

Set Tax Rates Command

Payload contains a cumulative array of all tax rates ever defined by the Tax Authority, along with their effective date and time.

NOTE:
This command is an *initialization* command during E-SDC's first communication with the Tax Service system. After that, the same command is used to *update* the existing tax rates.

The structure of a group is defined in section Tax Rates. The effective date can be in the past or in the future.

Tax rates with the future date should be stored in non-volatile memory and applied to start from their effective date. If more than one group has the same date, the one with higher revision (`groupId`) should be applied. This means that the next revision with the same effective date replaces the previous.

The payload of the command is structured as follows:

```
[
□{
□□"validFrom": "2022-01-15T06:27:58Z",
□□"groupId": 20,
□□"taxCategories": [
□□□{
□□□□"name": "ECAL",
□□□□"categoryType": 0,
□□□□"taxRates": [
□□□□□{
□□□□□□"rate": 10.0,
□□□□□□"label": "F"
□□□□□}
□□□□],
□□□□"orderId": 5
□□□},
□□□{
□□□□"name": "PB",
□□□□"categoryType": 2,
□□□□"taxRates": [
□□□□□{
□□□□□□"rate": 0.1,
□□□□□□"label": "P"
□□□□□}
□□□□],
□□□□"orderId": 6
□□□},
□□□{
□□□□"name": "STT",
□□□□"categoryType": 0,
□□□□"taxRates": [
□□□□□{
□□□□□□"rate": 1.0,
□□□□□□"label": "E"
□□□□□}
□□□□],
□□□□"orderId": 2
□□□},
□□□{
□□□□"name": "VAT",
□□□□"categoryType": 0,
□□□□"taxRates": [
□□□□□{
□□□□□□"rate": 9.0,
□□□□□□"label": "A"
□□□□□}
□□□□],
□□□□"orderId": 1
□□□}
□□]
□},
□{
□□"validFrom": "2021-09-30T22:00:00Z",
□□"groupId": 21,
□□"taxCategories": [
□□□{
□□□□"name": "ECAL",
□□□□"categoryType": 0,
□□□□"taxRates": [
□□□□□{
□□□□□□"rate": 10.0,
□□□□□□"label": "F"
□□□□□}
□□□□],
□□□□"orderId": 5
□□□},
□□□{
□□□□"name": "PB",
□□□□"categoryType": 2,
□□□□"taxRates": [
□□□□□{
□□□□□□"rate": 0.1,
□□□□□□"label": "P"
□□□□□}
□□□□],
□□□□"orderId": 6
□□□},
□□□{
□□□□"name": "STT",
□□□□"categoryType": 0,
□□□□"taxRates": [
□□□□□{
```

```
□□□□□□"rate": 1.0,
□□□□□□"label": "E"
□□□□□}
□□□□],
□□□□"orderId": 2
□□□},
□□□{
□□□□"name": "VAT",
□□□□"categoryType": 0,
□□□□"taxRates": [
□□□□□{
□□□□□□"rate": 9.0,
□□□□□□"label": "A"
□□□□□}
□□□□],
□□□□"orderId": 1
□□□}
□□]
□}
]
```

Set Verification URL Command

As a part of the invoice fiscalization, an E-SDC creates a unique URL for generating a QR code and validates an invoice. Verification URL is returned to the POS as a part of the Response. This command tells the E-SDC which URL will be used to Create Verification URL.

Payload is a URL of the server used to verify invoices. Detailed instructions for Verification URL creation are explained in Create Verification URL.

NOTE:
This command is an *initialization* command during E-SDC's first communication with the Tax Service system. After that, it is an *update* command for every next communication.

Example

https://sandbox.taxcore.online/v/?vl=

Forward Proof of Audit Command

Introduction

After the audit is executed successfully on the tax authority's system an E-SDC receives the proof-of-audit command. The E-SDC loads proof of audit and verifies if the format is valid. If the format is valid, proof of audit is sent to the Secure Element on the smart card (End Audit APDU command).

The payload is a byte array encoded as a base64 string.

If the format is invalid or the E-SDC and the Secure Element cannot process proof of audit for any reason, the E-SDC signals error message to the operator.

Example

```
0x0CEC878B2B7771B68D6D13EA3C1695A7EBCE395A7777940466EABB36ECA725300EE63E5BCF4CBDFC7C
```

Set TaxCore Configuration Command

This command's payload contains information about environment configuration. E-SDC stores this information locally and provides it to POS on demand. The same parameters can also be obtained from TaxCore.API. Payload is JSON structure as defined below.

NOTE:
This command is an *initialization* command during E-SDC's first communication with the Tax Service system. After that, it is an *update* command for every next communication.

Field	Description
organizationName	Name of the tax authority organization
serverTimeZone	Time zone of the server location
street	Tax authority street address
city	Tax authority city
country	Tax authority country
endpoints	List of available endpoints
environmentName	Name of the environment
logo	Link to the tax authority's official logo

ntpServer	NTP server used for time synchronization
supportedLanguages	List of language-culture strings supported by TaxCore

Example

```
{
  "organizationName": "Data Tech International",
  "serverTimeZone": "Fiji Standard Time",
  "street": "Kruzni put 7",
  "city": "Belgrade",
  "country": "RS",
  "endpoints": {
    "taxpayerAdminPortal": "https://tap.sandbox.taxcore.dti.rs:443/",
    "taxCoreApi": "https://api.sandbox.taxcore.dti.rs:443/",
    "vsdc": "https://vsdc.sandbox.taxcore.dti.rs:443/",
    "root": "https://frontendui.sandbox.taxcore.dti.rs:443/v/?vl="
  },
  "environmentName": "SANDBOX",
  "logo": "https://i.imgur.com:443/tg5ad6O.png?hash=1168891560",
  "ntpServer": "http://0.europe.pool.ntp.org:80/",
  "supportedLanguages": [
    "en-US",
    "sr-Cyrl-RS"
  ]
}
```

Forward Secure Element Directive Command

Introduction

Secure Element Directive payload is transmitted to the Secure Element Applet on the smart card using (Secure Element APDU command).

The payload is a byte array encoded as a base64 string.

Example

```
Xb/JzQSvncdsUPo/9U0y0ZELDS4exa+X6uPnGnQjzOBm1uJkJVg4wdut5hicqmwv82Laxuu9OiunaPpEM7R9
```

Processes

This section describes processes performed by an E-SDC.

It contains the following sections:

1. E SDC Initialization
   Prior to the first use, the E-SDC has to be initialized. E-SDC must have access to the Secure Element during the initialization process in order to establish a secure connection with the TaxCore.API to obtain a set of initialization commands. The initialization commands are explained in the section Commands.

2. Standard Operation
   This section contains a description of standard E-SDC operations.

3. Malfunctions and Non serviceable Devices
   If the Secure Element is damaged and its data cannot be restored from the card, but the E-SDC is operational, the tax authority system shall be able to dump data from the E-SDC device and upload the audit packages using the same application used to upload audit packages submitted by a taxpayer.

E-SDC Initialization

Prior to the first use, the E-SDC has to be initialized. E-SDC must have access to the Secure Element during the initialization process in order to establish a secure connection with the TaxCore.API to obtain a set of initialization commands. The initialization commands are explained in the section Commands.

For instructions on how to download the initialization commands, see Get Initialization Commands.

After processing the received initialization commands, the E-SDC must upload configuration commands results to TaxCore.API.

In case of a poor or no internet connection, configuration commands results can be uploaded via file-based communication as explained in section E-SDC Stores a Command Execution Result to the SD Card or USB Drive.

NOTE:
Initialization of E-SDC has to be performed each time a new smart card is inserted into the reader.

Standard Operation

This section contains a description of standard E-SDC operations.

1. Extracting Expiration Date from Digital Certificate
   Digital certificate exported using the *Export Certificate* APDU command (in DER format) contains the

certificate (secure element) expiration date.

2. Extracting Taxpayer Identification Number (TIN) from Digital Certificate-from-Digital-Certificate)
   Digital certificate exported using the *Export Certificate* APDU command (in DER format) contains taxpayer TIN and POS location (Shop or HQ Address that shall appear on the textual representation of the invoice).

3. Extracting UID from Digital Certificate
   Digital certificate exported using the *Export Certificate* APDU command (in DER format) contains the UID of the taxpayer's secure element.

4. Extracting Taxpayer Information from Digital Certificate
   Digital certificate exported using the *Export Certificate* APDU command (in DER format) contains taxpayer TIN, Business Name, Shop Name and POS location (Shop or HQ Address that will appear on the textual representation of the invoice).

5. E SDC Executes Commands
   E-SDC can receive Commands in two ways:

6. Sync Date and Time
   As an E-SDC is the source of date and time for the invoices, it is of the utmost importance to keep the device clock in sync.

7. Enter PIN to Unlock the Secure Element
   Before the Secure Element applet can be used, a valid PIN code must be supplied from the POS using the Ethernet connection. Once the E-SDC receives a PIN code, it will try to execute the *PIN Verify* APDU command.

8. Verify Secure Element Expiration Date and Time
   Each Secure Element is valid form 3 to 5 years (varies by Environment and local policies).

9. Fiscalization of an Invoice
   Invoice fiscalization is the main function of an E-SDC. Fiscalization is the process of handling an invoice request from an accredited invoicing system to produce fiscal invoices.

10. Audit Process
    An audit is a process of sequential transferring of audit packages from an E-SDC to the tax authority’s system and handling the response generated by the system for the specific device.

11. Notifications
    E-SDC device shall have an appropriate way to show the status of the device, information about the smart card and processes running on the E-SDC.

12.
    Switching Smart Cards During Operation
    During normal operation, taxpayers/cashiers might switch the smart card they are using for issuing fiscal invoices.

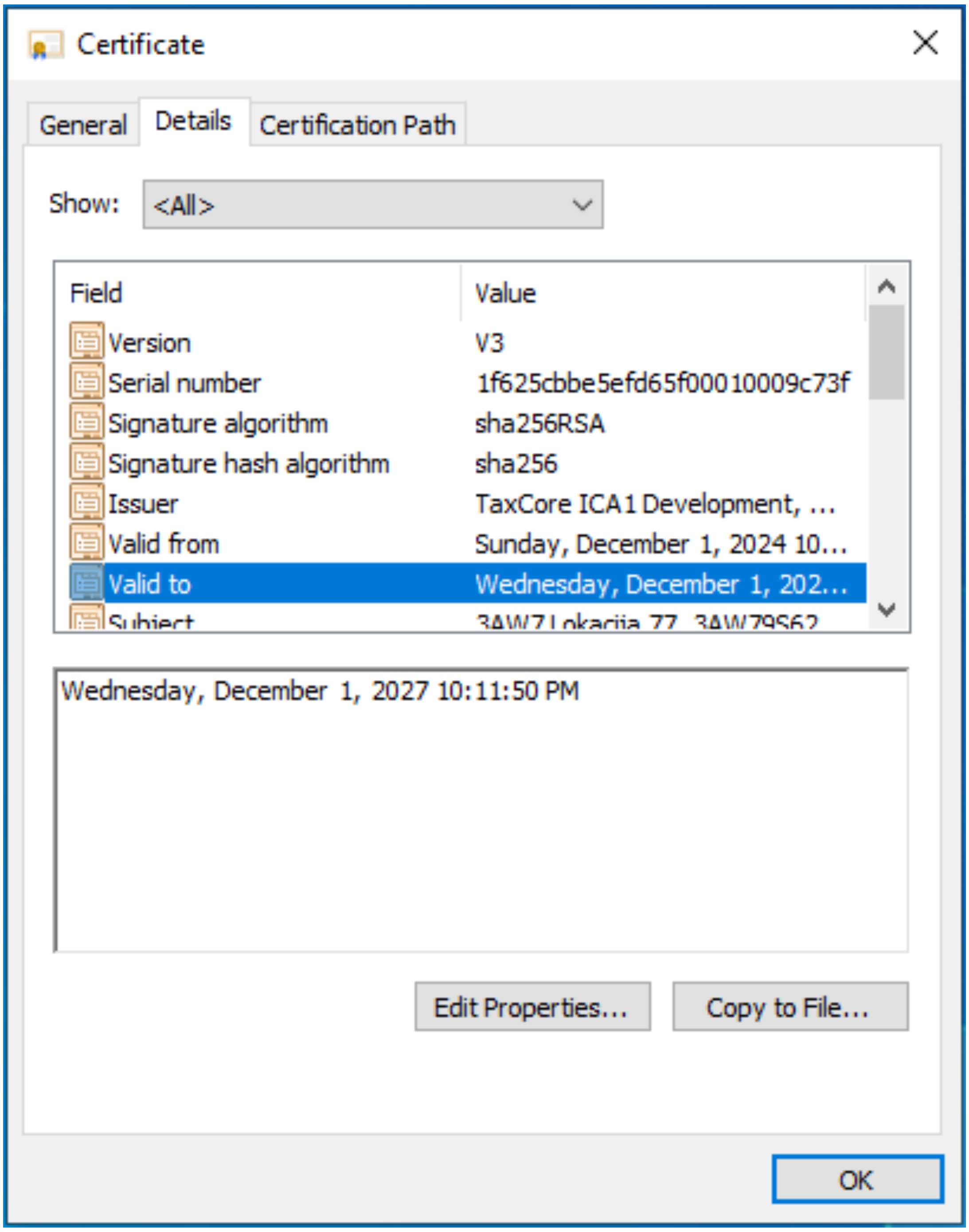
13.
    E SDC Logging
    E-SDC must keep a log about all required error events. It must log every error chronologically by local date and time (exact hour and minute).

Extracting Expiration Date from Digital Certificate

Digital certificate exported using the *Export Certificate* APDU command (in DER format) contains the certificate (secure element) expiration date.

Once the expiration date is extracted, it should be saved in **E-SDC's volatile memory**, until the smart card is removed/replaced or E-SDC is reset.

The date can be extracted from the certificate property **Valid to**



Field	Value
Version	V3
Serial number	1f625cbbe5efd65f00010009c73f
Signature algorithm	sha256RSA
Signature hash algorithm	sha256
Issuer	TaxCore ICA1 Development, ...
Valid from	Sunday, December 1, 2024 10...
Valid to	Wednesday, December 1, 202...
Subject	3AW7 Lokacija 77, 3AW79S62

Wednesday, December 1, 2027 10:11:50 PM

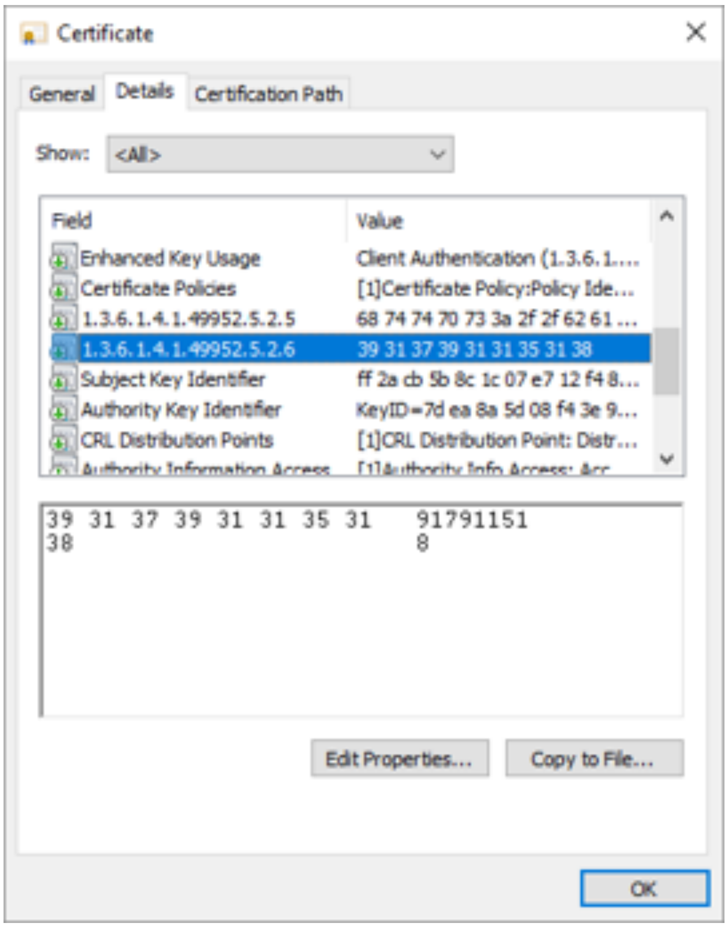
Extracting Taxpayer Identification Number (TIN) from Digital Certificate

Digital certificate exported using the *Export Certificate* APDU command (in DER format) contains taxpayer TIN and POS location (Shop or HQ Address that shall appear on the textual representation of the invoice).

TIN is stored in the digital certificate as an OID value. OID is dynamically created during a smart card personalization and depends on the target environment. The Test and Production environments will have different OIDs.

In order to use the same E-SDC with the Test and Production environments, the correct OID has to be constructed using the following procedure:

1. Get the certificate using the *Export Certificate* APDU command;
2. Read value of EnhancedKeyUsage (for example, 1.3.6.1.4.1.49952.5.2.3.3);
3. The fourth and the third integer to the right identify the environment;
4. Construct the OID that contains TIN, by replacing stars with the numbers using the following pattern - 1.3.6.1.4.1.49952...6;
5. For this example, resulting OID will be 1.3.6.1.4.1.49952.5.2.6;
6. Read the value of resulting OID containing Taxpayer TIN.



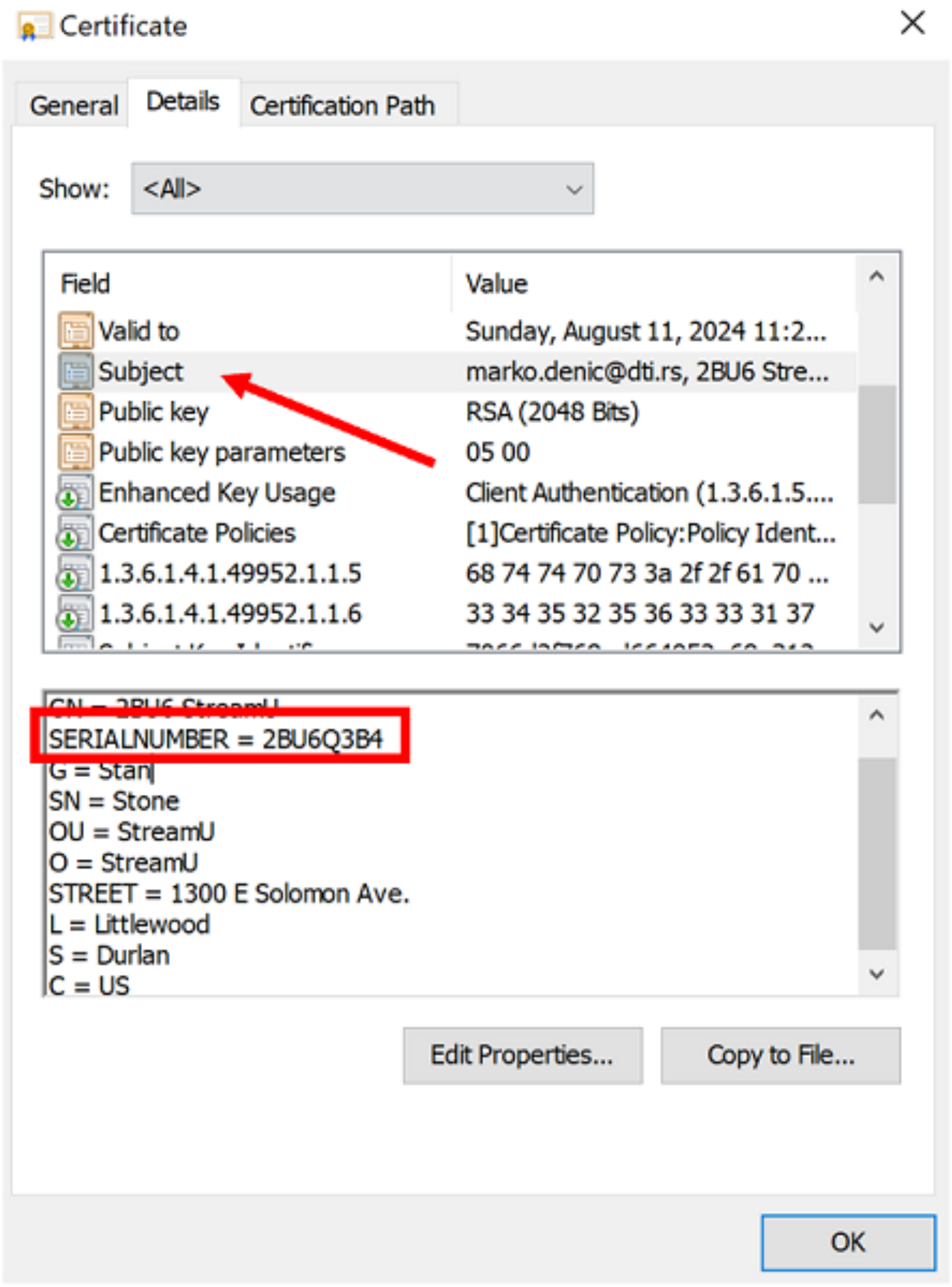
Field	Value
Enhanced Key Usage	Client Authentication (1.3.6.1....
Certificate Policies	[1]Certificate Policy:Policy Ide...
1.3.6.1.4.1.49952.5.2.5	68 74 74 70 73 3a 2f 2f 62 61 ...
1.3.6.1.4.1.49952.5.2.6	39 31 37 39 31 31 35 31 38
Subject Key Identifier	ff 2a cb 5b 8c 1c 07 e7 12 f4 8...
Authority Key Identifier	KeyID=7d ea 8a 5d 08 f4 3e 9...
CRL Distribution Points	[1]CRL Distribution Point: Distr...
Authority Information Access	[1]Authority Info Access: Acc

```
39 31 37 39 31 31 35 31    91791151
38                         8
```

Extracting UID from Digital Certificate

Digital certificate exported using the *Export Certificate* APDU command (in DER format) contains the UID of the taxpayer's secure element.

UID is stored in the digital certificate as an **SERIALNUMBER** value in the certificate **Subject** field.



Field	Value
Valid to	Sunday, August 11, 2024 11:2...
Subject	marko.denic@dti.rs, 2BU6 Stre...
Public key	RSA (2048 Bits)
Public key parameters	05 00
Enhanced Key Usage	Client Authentication (1.3.6.1.5....
Certificate Policies	[1]Certificate Policy:Policy Ident...
1.3.6.1.4.1.49952.1.1.5	68 74 74 70 73 3a 2f 2f 61 70 ...
1.3.6.1.4.1.49952.1.1.6	33 34 35 32 35 36 33 33 31 37

```
CN = 2BU6 StreamU
SERIALNUMBER = 2BU6Q3B4
G = Stan
SN = Stone
OU = StreamU
O = StreamU
STREET = 1300 E Solomon Ave.
L = Littlewood
S = Durlan
C = US
```

Extracting Taxpayer Information from Digital Certificate

Digital certificate exported using the *Export Certificate* APDU command (in DER format) contains taxpayer TIN, Business Name, Shop Name and POS location (Shop or HQ Address that will appear on the textual representation of the invoice).

Subject Field of the certificate contains the following information:

CN = Certificate Common Name - first 4 characters of the secure element UID and the Shop name

SERIALNUMBER = UID if the taxpayer's secure element

G = Authorized Person first name

SN = Authorized Person first name

OU = Shop name

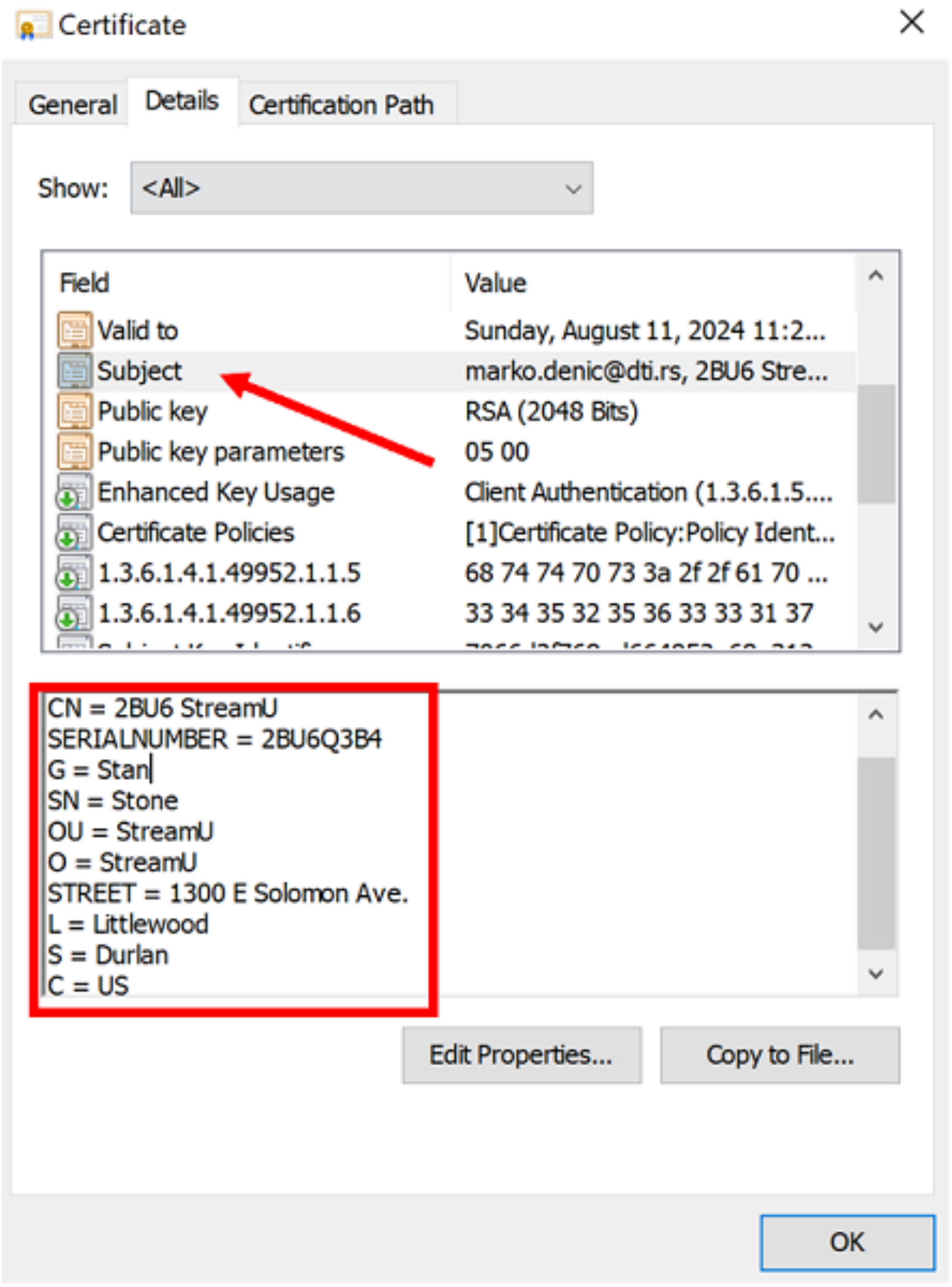
O = Business name

STREET = Physical street address

L = City/town

S = State, District or Region

C = Country



Certificate

Show: <All>

Field	Value
Valid to	Sunday, August 11, 2024 11:2...
Subject	marko.denic@dti.rs, 2BU6 Stre...
Public key	RSA (2048 Bits)
Public key parameters	05 00
Enhanced Key Usage	Client Authentication (1.3.6.1.5....
Certificate Policies	[1]Certificate Policy:Policy Ident...
1.3.6.1.4.1.49952.1.1.5	68 74 74 70 73 3a 2f 2f 61 70 ...
1.3.6.1.4.1.49952.1.1.6	33 34 35 32 35 36 33 33 31 37

```
CN = 2BU6 StreamU
SERIALNUMBER = 2BU6Q3B4
G = Stan
SN = Stone
OU = StreamU
O = StreamU
STREET = 1300 E Solomon Ave.
L = Littlewood
S = Durlan
C = US
```

E-SDC Executes Commands

E-SDC can receive Commands in two ways:

- as a response to some TaxCore.API service calls (Notify Online Status and Submit Audit Package)
- via a external storage units

Processing commands depends on the command type. The process of execution for each command is explained in the Commands section.

After the execution of the commands, in case the E-SDC is online, it notifies TaxCore.API about the execution success as explained in Notify Command Processed.

Sync Date and Time

As an E-SDC is the source of date and time for the invoices, it is of the utmost importance to keep the device clock in sync.

If the internet connection is available, the E-SDC shall sync time with the recommended NTP service at least once every 48h.

If the E-SDC does not support online or semi-connected operation modes, the manufacturer shall provide and document a simple way to check, set and keep date and time in sync on the E-SDC.

Enter PIN to Unlock the Secure Element

Before the Secure Element applet can be used, a valid PIN code must be supplied from the POS using the Ethernet connection. Once the E-SDC receives a PIN code, it will try to execute the *PIN Verify* APDU command.

NOTE:
Depending on the provided PIN, the Secure Element will remain either unlocked for further use or locked until a valid PIN is entered. This means that the E-SDC should not execute the PIN Verify APDU command again as long as the communication session between the E-SDC and the smart card is open.

E-SDC will send a response to the POS based on the result of the PIN Verify command execution.

It is important to note the Secure Element interprets data as byte containing digits, so the E-SDC must perform appropriate conversion before data is sent to the Secure Element. For example, if a PIN transmitted from a POS is “2017” (0x32 0x30 0x31 0x37 in ASCII hexadecimal representation), data sent to the SE shall be 0x02 0x00 0x01 0x07.

Verify Secure Element Expiration Date and Time

Introduction

Each Secure Element is valid form 3 to 5 years (varies by Environment and local policies).

Any invoice fiscalized using expired secure element will be automatically marked as invalid.

How to obtain digital certificate

E-SDC must obtain Secure Element expiration date from digital certificate returned from Secure Element using Export Certificate APDU command in DER format.

This operation must be executed whenever smartcard is inserted into reader or E-SDC is switched on.

What E-SDC must do if digital certificate has expired?

E-SDC must check current date and time against expiration date and time **BEFORE** invoice is signed by Secure Element. If current date and time is greater than expiration date and time of the digital certificate E-SDC must stop fiscalizing any invoices.

Fiscalization of an Invoice

Introduction

Invoice fiscalization is the main function of an E-SDC. Fiscalization is the process of handling an invoice request from an accredited invoicing system to produce fiscal invoices.

Process

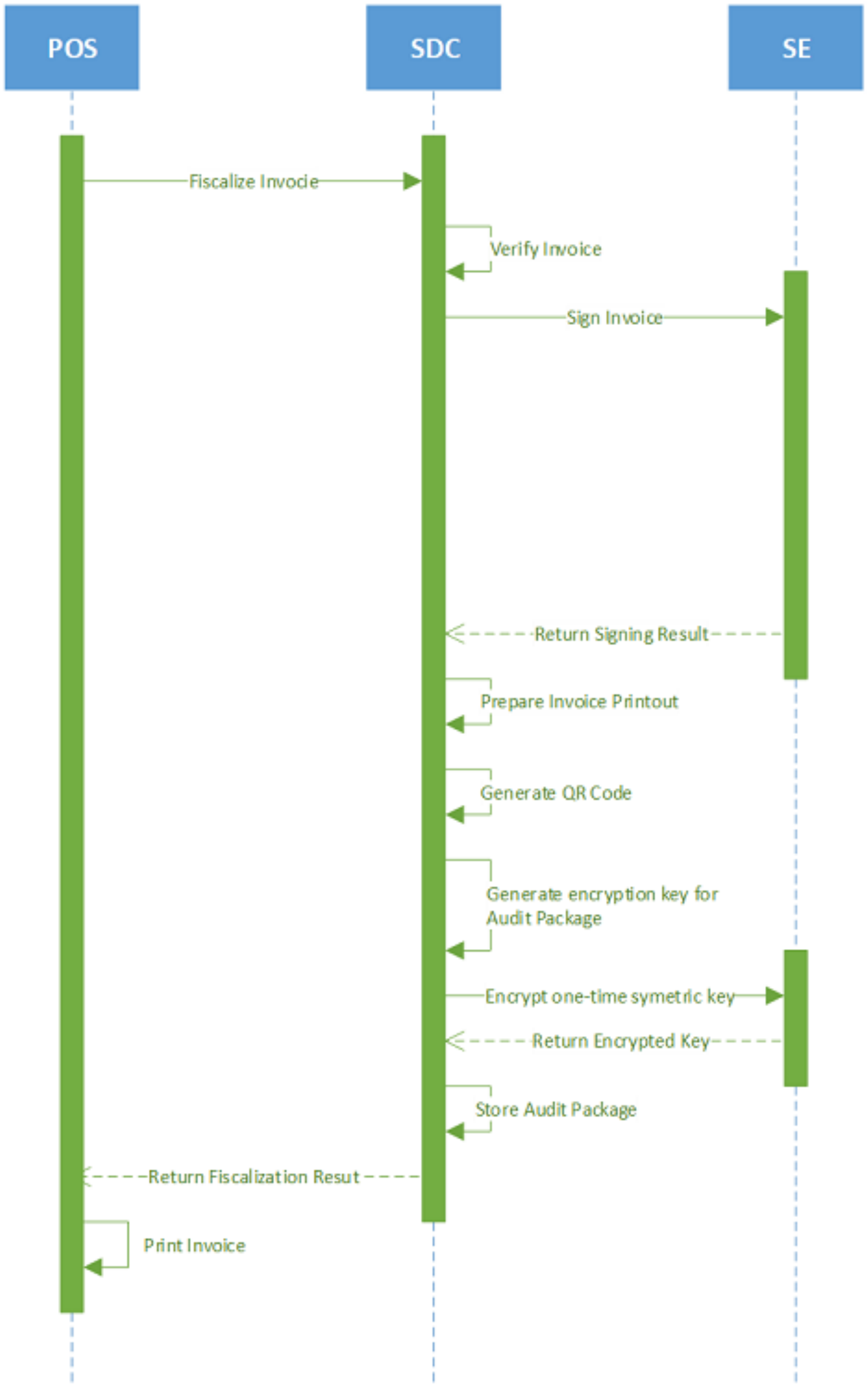
The following steps are executed by the E-SDC once an invoice request data is received from an Accredited POS:

1. POS generates a request data and sends it as an invoice request to the E-SDC;
2. E-SDC verifies the format and content of the invoice request;
3. E-SDC verifies the current E-SDC date and time value is smaller than the smart card certificate expiration date;
4. E-SDC determines which tax rate group to use based on the value of invoice type, ReferentDocumentNumber and ReferentDocumentDT fields;
5. E-SDC calculates taxes based on the selected tax rates group;
6.

E-SDC sends the invoice data to the Secure Element for fiscalization providing the current date and time and PIN code/password if required;

7. Secure element signs the invoice and returns the data to the E-SDC;
8. E-SDC produces a journal – a textual representation of an invoice;
9. E-SDC generates a verification URL;
10. [optionally] E-SDC creates a QR Code – a graphical representation of a verification URL;
11. E-SDC creates an invoice with all mandatory elements (receipt data, previously generated signature, verification URL and journal), generates a one-time key and encrypts the invoice using a symmetric algorithm. The E-SDC encrypts a one-time symmetric key using the tax authority's system public key and adds it to the package so the tax authority's system decrypts the symmetric key and access the package content once it arrives in the Service's system.
12. E-SDC returns a response to the POS and optionally generates journal data.

The process is illustrated in the figure below.



Fiscalization of An Invoice – Image of the fiscalization process

Content

Calculate Taxes

Introduction

Taxes are calculated by an E-SDC after a POS has sent a valid request. The tax amount for particular items on an invoice is defined by the tax labels associated with an item.

Process of a tax calculation depends on:

- Invoice and Transaction Type
- the tax rates for each label associated with an item on an invoice
- the Type value of tax category to which the label belongs

A POS sends an invoice fiscalization request with the line items. Items are sent with the total amounts (taxes included) and zero or more tax labels associated with them, which participated in the total price calculation.

How To Determine Which Tax Rate Group to Use

By default, E-SDC must use current `TaxRateGroup` based on current date and time of its clock to calculate taxes for each invoice.

In case of Copies and Refunds following rules are applied:

- If `referentDocumentNumber` is specified and `referentDocumentDT` is ommited or blank applicable `TaxRateGroup` must be determined based on current date and time of its clock and used to calculate taxes
- If both `referentDocumentDT` and `referentDocumentNumber` are specified, `TaxRateGroup` shall be determined based on value of `referentDocumentDT`

Algorithm

In order to calculate a tax, the following algorithm shall be implemented:

1. Determine which Tax Rate Group to use (see above).
2. Make an array of distinct tax labels associated with the items in the POS request (e.g. A, B, C, F, ...).
3. Calculate the tax amount for each individual label in the array:
    - Iterate through all items in the POS request
    - For each item, calculate tax amounts. One item has one or more tax labels, and each label represents a tax amount. Each tax amount is a part of an item’s total price. These tax amounts are calculated as follows:
        - If an item has a label from the **amount-on-quantity** category applied, subtract the tax rate amount for that label, multiplied with quantity, from the item total price. The resulting amount (the remainder), is used in all further calculation steps instead of item total amount.
        - If none of the labels’ tax category type is ***tax-on-total*** (category 1):
            - Tax amount for one label is:

$$\frac{item\ total\ amount\ *\ label\ rate}{(100 + \Sigma(all\ tax - on - net\ rates\ on\ item))}$$

Example 1: An item has a total price of $10 and applied labels: A(5%) and B(6%).

$$A = \frac{10\$ * 5}{(100 + \Sigma(5+6))} \quad B = \frac{10\$ * 6}{(100 + \Sigma(5+6))}$$

Tax amount for label A=$0.4505 and for label B=$0.5405.

- If any of the labels' tax category is ***tax-on-total*** (category 1):
  - Tax amount for every label whose category type is ***tax-on-total*** (category 1) is:

$$\frac{item\,total\,amount}{(1 + \Sigma(all\ tax - on - total\ rates)/100)} * \frac{label\ rate}{100}$$

  - Tax amount for every other label from category 0 is:

$$\frac{item\,total\,amount}{(1 + \Sigma(all\ tax - on - total\ rates)/100)} * \frac{label\ rate}{(100 + \Sigma(all\ tax - on - net\ rates\ on\ item))}$$

Example 2: Item has a total price $10 and applied labels: A(5% tax-on-net), B(6% tax-on-net), C(3% tax-on-total) and F(4% tax-on-total).

$$A = \frac{10\$}{(1 + \Sigma(3+4)/100)} * \frac{5}{(100 + \Sigma(5+6))}$$

$$B = \frac{10\$}{(1 + \Sigma(3+4)/100)} * \frac{6}{(100 + \Sigma(5+6))}$$

$$C = \frac{10\$}{(1 + \Sigma(3+4)/100)} * \frac{3}{100}$$

$$F = \frac{10\$}{(1 + \Sigma(3+4)/100)} * \frac{4}{100}$$

Tax amount for label A=$0.4210 , for label B=$0.5052 , for label C=$0.2804 and for label F=$0.3738.

- Summarize calculated tax amounts per label for items as the total amount sum for that label.

  Example 3: the request contains two items from Example 1 and Example 2, the total sums for labels are: A=$0.8715 , B=$1.0457 , C=$0.2804 and F=$0.3738.

- Summarize fixed tax amounts per label for items (each multiplied with quantity) as the respective total amount sum for that label.

  Example 4: An item has quantity 2 with a total price of $10 and applied labels: A(5% tax-on-net) and E($0.10 fixed tax). The total sums for labels are: A=$0.4667 and E=$0.2000

  Example 5: An item has quantity 2 with total price 10$ and applied labels: A(5% tax-on-net), C(3%

tax-on-total) and E(0.10$ fixed tax). The total sums for labels are: A=$0.4531, C=$0.2854 and E=$0.2000.

Example 6: the request contains two items: one with a total price of $5 and quantity 1, and another with a price of $10 and quantity 2. Both have applied label E(0.10$ fixed tax). The total sum for label E=$0.3000.

- Each of these summarized amounts per label represents one tax item in the array `taxItems` in [Invoice Response](), along with other properties related to the category for this label.

4. After all of the items have been processed, calculate the tax amount for all tax categories found in the request. One tax category can consist of one or more tax labels (e.g. A, B...). The tax amount for a tax category is a sum of all label tax amounts related to the category. These tax amounts for the category are displayed on the invoice journal.

   Example 7: The request contains two items from Example 1 and Example 2. Labels A and B are VAT category, C is STT category and F is ET category. Total VAT=$1.9172, STT=$0.2804 and ET=$0.3738.
5. Once the Tax calculation is completed, assign `GroupId` of the tax rate group to the field `TaxGroupRevision` of `InvoiceResult`.

Rounding

E-SDC shall round all amounts to 4 decimal places using the half-round up method.

Examples:

3.44445555666 → 3.4445

3.4440012345 → 3.4440

3.44466012345 → 3.4447

3.444116012345 → 3.4441

Create Verification URL

Verification URL is created based on values submitted by a POS to an E-SDC and values returned to the E-SDC from APDU commands as follows:

1. Byte array is created:

Start	Bytes	Invoice Field	Is an invoice field	Description
0	1	version	Yes	Current version is 0x03
1	8	requestedBy	Yes	UID, ASCII encoding (e.g. JKGB3K14)
9	8	signedBy	Yes	UID, ASCII encoding (e.g. JKGB3K14)
17	4	totalCounter	Yes	Int32 Little Endian
21	4	transactionTypeCou	Yes	Int32 Little Endian
25	8	totalAmount	Yes	totalInvoiceAmount * 10000 as Uint64 bit Little Endian
33	8	dateAndTime	Yes	Unix Timestamp (number of milliseconds), 64bit unsigned integer Big Endian
41	1	invoiceType	Yes	0x00 (Normal), 0x01 (Pro Forma), 0x02 (Copy), 0x03(Training),0x04(A
42	1	transactionType	Yes	0x00 (Sale), 0x01 (Refund)
43	1	N/A	No	Buyer ID length in bytes
44	0-20	buyerId	Yes	Unicode Encoding
44-64	256 or 512	encryptedInternalD	Yes	Encrypted Internal Data received from SE after Invoice Sign APDU command, 256 or 512 bytes long
300-320 or **556-576**	256	signature	Yes	Signature received from SE after Invoice Sign APDU

				command, 256 bytes long
556-576 or **812-832**	16	N/A	No	MD5 hash of all previous bytes

2. Created byte array is encoded as base64 string, which is additionally encoded, to comply with the URL standards.
3. Encoded string is appended to the verification URL received from the Set Verification URL Command.

NOTE:
Values for `dateAndTime` and all other fields must match the values submitted to the Secure Element for digital signing (see *Sign Invoice* in Fiscalization), as well as the values in the audit package (see Create Invoice) .

Create a QR Code

QR code contains a Verification URL that is described created in the section Create Verification URL.

It is the most convenient way of exposing the Verification URL because it enables customers to easily scan their fiscal invoices using a QR code reader.

How to create a QR code

Base64 encoded string is created from GIF image bytes and attached to the Invoice Response

Important parameters for creating a QR code:

- Minimal size = 40x40mm
- ErrorCorrectionLevel = L
- FixedModuleSize = 4
- QuietZoneModules = Zero
- BlackAndWhite
- ImageFormat = Gif

For more information about using QR codes in the TaxCore system, see QR Code

Create a Textual Representation of an Invoice

A textual representation of a Receipt shall be created as described in the chapter Anatomy of a Fiscal Receipt. One row on a receipt is 40 characters long to fit 2.25 inch / 58 mm wide paper roll commonly used in thermal printers.

SDC Date and Time field printed on a journal (textual representation of an invoice) generated by E-SDC are **locally time-based**.

Any amount shall be rounded to 2 (two) decimal places using the half-round up method only on the textual representation of an invoice.

NOTE:
Although the textual representation of an invoice (journal) can optionally be omitted from the E-SDC's response to POS, it **must be submitted to the tax authority** as part of the audit package.

1. Localization of textual representation of the invoice
   E-SDCs are generally built to work in multiple environments. As part of implementation roadmap you may decide to prepare your product for multiple markets or to target one market only.

2. Mapping Digital Certificate Subject Parameters to Invoice Fields
   Digital certificate exported using the *Export Certificate* APDU command (in DER format) contains taxpayer TIN and POS location (Shop or HQ Address that shall appear on the textual representation of the invoice).

Localization of textual representation of the invoice

Introduction

E-SDCs are generally built to work in multiple environments. As part of implementation roadmap you may decide to prepare your product for multiple markets or to target one market only.

Localization is optional

If you decide to support only one market E-SDC may support only one language. For example, Implementation for Fiji may support English language only.

What to do if you want to support multiple

languages?

- Textual representation must adhere to general instructions outlined in Anatomy of a Fiscal Receipt.
- All Invoice and transaction types abbreviations must be localized. For example **NS** (Normal Sale) in English is localized as **ПП** (Промет Продаја) in Serbian Cyrilic.
- All labels on invoice must be translated
- Languages supported by your E-SDC implementation **AND** TaxCore.API must be contained in the Result of the Get Status service.
  - in case your E-SDC supports en-US, fr-FR and sr-Latn-RS and TaxCore.API supports only fr-FR E-SDC must report fr-FR as the only supoprted language
  - in case your E-SDC supports en-US only and TaxCore.API supports only fr-FR E-SDC must return configuration error and stop any initialization.
- Content generated by POS or Secure Element should not be modified or localized (i.e. item names)
- If your E-SDC and TaxCore.API supports multiple languages POS may decide language of generated textual representation of invoice using request HTTP headers when invoking Create Invoice endpoint.

Accreditation

Accreditation for specific jurisdiction may require E-SDC to support specific language and culture. For Example, E-SDCs for Serbia requires support for sr-Cyrl-RS.

Mapping Digital Certificate Subject Parameters to Invoice Fields

Digital certificate exported using the *Export Certificate* APDU command (in DER format) contains taxpayer TIN and POS location (Shop or HQ Address that shall appear on the textual representation of the invoice).

This example shows the mapping between a subject name/value pairs and invoice fields.

Subject Field parameters with examples:

CN = P22V International Trek Center

SERIALNUMBER = P22VC8VR

G = Albert

SN = Mungin

OU = International Trek Center

O = International Trek Center

STREET = 8844 Garcia

L = West Covina

S = California

C = US

Invoice Field	Subject Parameter Name	Note
TIN	N/A	obtained by OID as explained in Extracting Taxpayer Identification Number from Digital Certificate
Business Name	O	Legal name under which the business operates - see Extracting Taxpayer Information from Digital Certificate
Shop Name	OU	It may be the same as Business Name if the Company HQ and sales location are the same. - see Extracting Taxpayer Information from Digital Certificate
Address	STREET	Street name and number - see Extracting Taxpayer Information from Digital Certificate
Location	L	City or town - see Extracting Taxpayer Information from Digital Certificate
State	S	State, District or Region - see Extracting Taxpayer Information from Digital Certificate
Country	C	ISO 2-letter Country Code. Optional field on the textual representation of the invoice - see Extracting Taxpayer Information from Digital Certificate

Creating an Audit Package

Once an invoice is created (`InvoiceRequest` and `InvoiceResult`) the E-SDC is ready to create an audit package and store it in the non-volatile memory. In order to achieve that, follow these steps:

1. Convert `sdcDateTime` data to UTC;
2. Generate a random one-time symmetric key for AES256;
    - o KeySize = 256
    - o Padding = PaddingMode.PKCS7
    - o Mode = CBC
    - o BlockSize = 128
    - o IV = 16bytes
    - o Key =32bytes
3. Encrypt Audit Data as JSON string using the one-time key;
4. Convert the encrypted invoice to base64 string and store it in the Payload field of Audit package;
5. Get the TaxCore Public key using *Export TaxCore Public Key* APDU command).
6. Encrypt the one-time key, using RSA encryption (padding PKCS1 (fOAEP: false)), with TaxCore public key, convert it to base64 string and store it in the Key field;
7. Encrypt Initialization Vector (IV), using RSA encryption (padding PKCS1 (fOAEP: false)), with TaxCore public key, convert it to base64 string and store it in the IV field;
8. Save the Audits as an Audit Package file, named as `{UID}-{UID}-{Ordinal_Number}.json`;
9. (Optionally) Generate a QR code, and attach it to `InvoiceResult` (make sure that the QR code is not stored in the Audit Package);
10. Return `InvoiceResult` to the POS;
11. If the internet connection is available try to send the Audit Data to TaxCore.API as explained in the section Remote Audit;

NOTE:
After submitting an audit package to TaxCore.API, if status 4 is received back (see Submit Audit package) , the E-SDC should immediately delete that audit package from the its local storage. If the TaxCore.API returns status 1, the E-SDC should try to resubmit an audit package. If any other status is received (other that 1 or 4), the audit package must not be deleted, and the E-SDC should not try to resubmit that audit package to TaxCore.API.

Audit Process

Introduction

An audit is a process of sequential transferring of audit packages from an E-SDC to the tax authority’s system and handling the response generated by the system for the specific device.

There are two specific scenarios: **Remote Audit** and **Local Audit**.

NOTE:
Basic rules and processes described in this section apply to both scenarios. Details are explained in separate sections - see Remote Audit and Local Audit.

Depending on the scenario, an audit may be triggered periodically, if one or more invoices are created, or after the insertion of an external memory device into an E-SDC.

An audit is always an asynchronous process. Depending on the amount of data and means of communication, it can take from less than a second to a couple of hours.

Once the E-SDC receives a response from the Secure Element (signed invoice), it must be encrypted and stored in E-SDC's non-volatile memory.

An E-SDC device must be fully functional during an audit. The POS must be able to sign new invoices as long as the Secure Element permits. There must be a mechanism in place that is responsible for the continuous operation of the Secure Element and E-SDC while audit packages are being transmitted to the tax authority’s system or an external memory unit.

Remote Audit

Remote audit is the process of transferring data to the tax authority’s system using an internet connection. It is the most common way to perform audits for any device with a stable internet connection.

An E-SDC checks if TaxCore.API is reachable. If TaxCore.API is reachable, the E-SDC authenticates the tax authority’s system by using a server-side certificate installed on the TaxCore.API endpoint, enabling HTTPS protocol. The tax authority’s system authenticates the E-SDC using a digital certificate issued on the Secure Element and issues a token for that session.

The E-SDC starts sending audit packages, performing a series of audits until all the data stored on its non-volatile memory is audited.

A Remote audit is not the only audit option for E-SDC. If the network connection is not available due to the interruption of the service or a missing GPRS modem or network card, E-SDC is able to perform a Local Audit.

Remote audit step-by-step:

1. E-SDC submits an audit package by invoking TaxCore.API service Submit Audit Package
2. Tax authority's system verifies the data and returns a response containing:
   - audit package status
   - a set of Commands (including the Proof of Audit command)

Local Audit

Local audit initiated by a taxpayer is a common scenario for devices that lack the ability to connect to the internet due to the technical limitations of the devices or limited infrastructure.

Unlike in Remote Audit process, during the Local Audit, the E-SDC doesn't submit the ARP file and audit packages to TaxCore.API. Instead, those files are copied to an SD Card or a USB Flash Drive.

Local audit step-by-step:

1. An audit is initiated by inserting an SD card or a USB Flash drive into an E-SDC device.
2. E-SDC signals the beginning of the audit to the Secure Element (invokes *Start Audit* APDU command);
3. E-SDC copies audit packages to external memory unit (e.g. SD card, USB flash drive), starting with the oldest unaudited package, in a piecemeal fashion, as described in E-SDC Stores Audit Files on SD Card or USB Drive.

NOTE:
Audit packages remain stored in E-SDC's local memory until it receives a Proof-of-Audit command from TaxCore.API

4. The Secure Element returns ARP (260 bytes) to the E-SDC;

NOTE:
When performing Local Audit, the ARP file should be generated and saved each time when at least one audit package is submitted to the tax authority’s system, by using an external memory unit (see File-based communication).

5. An operator uploads the audit packages and the ARP file to the Tax Authority's system (by using the Taxpayer Administration Portal or the tax authority's Back Office Portal).
6. After the successful upload, a set of Commands (including the Proof of Audit command) are generated by the tax authority's system.

7. An operator downloads the commands to an external memory unit (by using the Taxpayer Administration Portal or the tax authority's Back Office Portal).
8. An operator inserts the external memory unit with the commands file into an E-SDC which processes the commands as described in E-SDC Executes Commands, i.e. passes the payload to the *End Audit* APDU command;

NOTE:
The Proof of Audit command might be missing due to many reasons, such as: not all packages have been submitted, the tax authority system is experiencing a delay in data processing... Perhaps, the Proof of Audit command for a specific local audit might never be returned.

9. The Secure Element resets its current unaudited amount to 0.00 and returns the OK message to the E-SDC;
10. The E-SDC generates the `Commands Execution Results` structure (as described in Commands) and saves it to a file on the external media, as described in E-SDC stores a command execution result to the SD card or USB drive.
11. The operator uploads the `Commands Execution Results` file to the Tax Authority's system (by using the Taxpayer Administration Portal or the tax authority's Back Office Portal).

Proof of Audit

Introduction

Proof-of-Audit (POA) is generated by the tax authority’s system once all expected audit packages have been received and securely stored on the tax authority’s system.

Even if, due to the failure of the EFD component or some other reason, the taxpayer is unable to send one or more audit packages, the Tax Authority still has the option to issue a Proof-of-Audit if it can determine the tax liability for that secure element.

A POA cycle begins with E-SDC initiating the *Start Audit APDU command* (**Audit Start**). A POA cycle finishes with the Tax Authority's system receiving a confirmation that the secure element has successfully executed the issued Proof-of-Audit command (**Audit End**).

NOTE:
This article describes the default operation mode of the *Proof of Audit Service*. However, upon the Tax Authority’s decision, the service can use different strategies for issuing a proof-of-audit. For the currently applicable non-

standard issuance strategies, see Currently Applicable Non-Standard Strategy for Issuing Proof of Audit.

There are two scenarios for obtaining a Proof-of-Audit:

- **As part of the Local Audit process**
- **Via direct communication with TaxCore.API as described below**

1. E-SDC signals the beginning of the audit to the Secure Element (invokes *Start Audit* APDU command);
2. The Secure Element returns ARP (260 bytes) to the E-SDC;
3. E-SDC starts the audit by sending audit data (over HTTPS). ARP is submitted to the tax authority's system using the same communication channel;
4. If the request is correct, the system returns the HTTP status code 200 (OK);
5. Tax authority's system generates a set of Commands (including the Proof of Audit command);
6. E-SDC submits a commands request to the tax authority's system via one of the following channels:
   - by invoking TaxCore.API service Notify Online Status
   - by invoking TaxCore.API service Submit Audit Package
7. Tax authority's system returns the commands to the E-SDC as a response;
8. E-SDC processes the commands as described in E-SDC Executes Commands, i.e. passes the payload to the *End Audit* APDU command;
9. The Secure Element resets its current unaudited amount to 0.00 and returns an OK message to the E-SDC;
10. E-SDC reports the results of the commands' execution to TaxCore.API as described in Notify Commands Processed

NOTE:
Regardless of which scenario was used to initiate the POA cycle, once the commands (containing the Proof-of-Audit command) have been generated by the tax authority's system, the cycle can be completed by using the other scenario.

POA cycle frequency

Audit Start should be initiated periodically, and the length of the period between two Audit Starts should be set dynamically. In regular conditions, there is no reason to initiate an Audit more often than every 30 minutes; although this can be extended to a period of a couple of hours as well (depending on the turnover). There is no need to initiate unnecessary Audits if the secure element(s) does not register sale amounts that will cause it to cross its assigned limit.

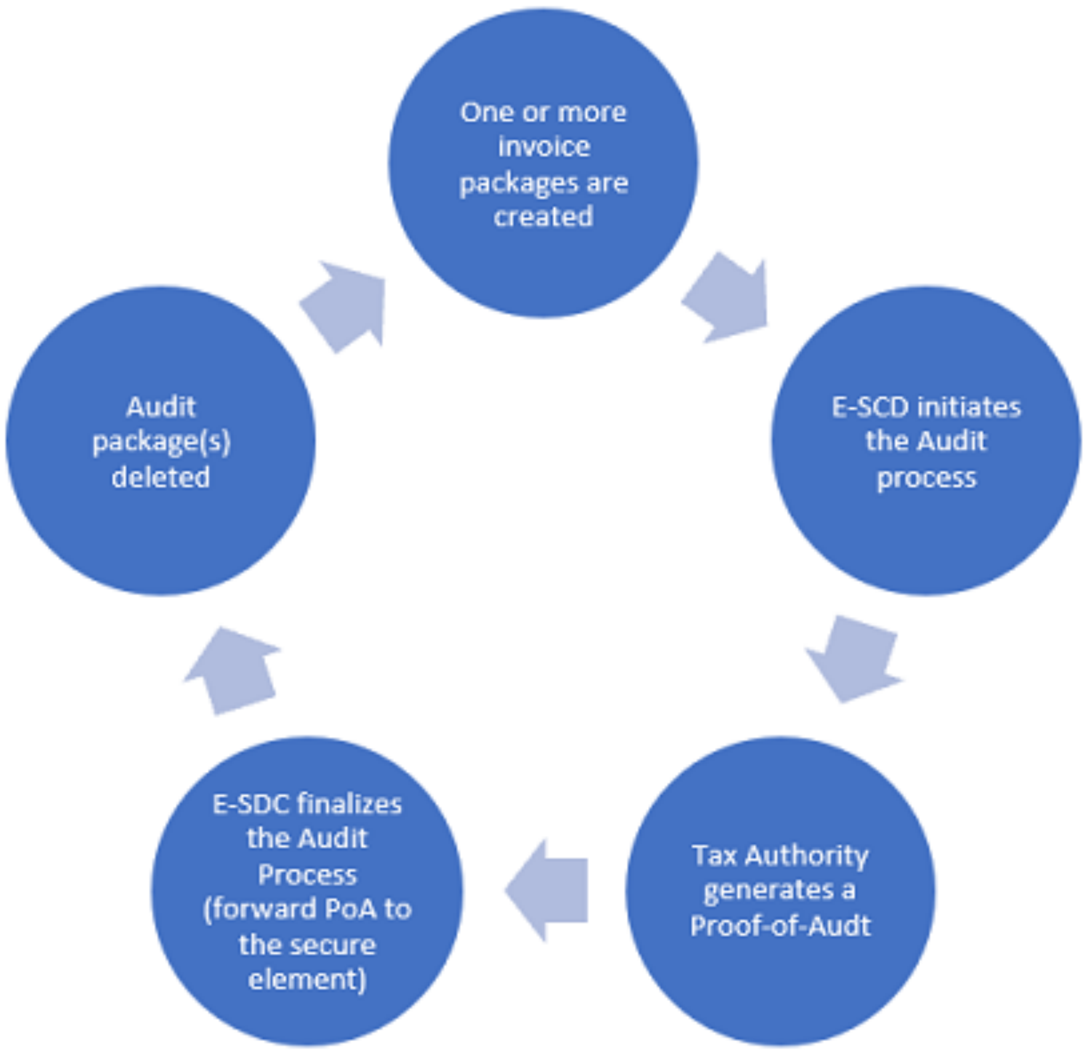
However, if the secure element is reaching its limit, the Audit should be initiated immediately and the period between two Audit Starts should be shortened to 10 minutes (unless there were no new invoices created in that period).

NOTE:
Although the Audit Start should be initiated periodically, audit packages should be submitted continuously, as soon as they are created (regardless of the Audit Start and Audit End).

The СУФ Развој system needs about 10 minutes to process submitted data and generate a Proof-of-Audit command, provided that there are no missing audit packages for that Audit.

Once the Secure Element receives a valid Proof-of-Audit, the E-SDC can delete the audit packages included in that Audit.

NOTE:
E-SDC must store the SDC number of the last invoice created before an Audit Start (the last invoice included in that Audit) so it would know which invoices to delete upon receiving a valid Proof-of-Audit.


One or more invoice packages are created
E-SCD initiates the Audit process
Tax Authority generates a Proof-of-Audt
E-SDC finalizes the Audit Process (forward PoA to the secure element)
Audit package(s) deleted

Be mindful of these cases

Sometimes, audit packages can arrive in СУФ Развој database after the Start Audit command - in that case, the system will again need at least 10 minutes after the arrival of the last audit package to generate the End Audit command (**see case 2a below**).

If two Audit Starts are initiated before an Audit End command is generated, the Proof-of-Audit for the first Audit Start will not be valid. Only the next Proof-of-Audit (covering both Audit Starts) will be valid (**see Case 3 below**). Moreover, the Proof-of-Audit for the first Audit Start will be replaced by the next Proof-of-Audit, so E-SDC will receive only the latest Proof-of-Audit. But if E-SDC happens to receive the first Proof-of-Audit, it will be invalid and rejected by the Secure Element.

If one or more audit packages issued by the same secure element (same UID) do not arrive in the database, the End Audit command (Proof-of-Audit) will never be generated.

Audit cycle cases

This means that there are three possible scenarios for completing the Audit cycle:

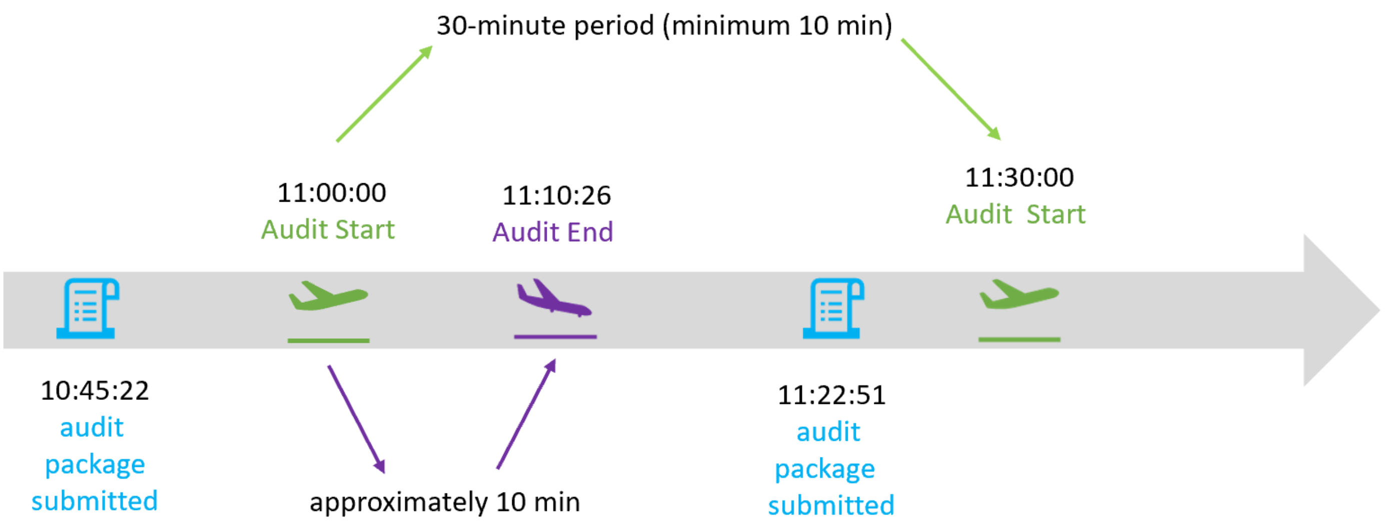
1. One audit package is created and one Audit Start is initiated between completing two Audit Ends (two Proof-of-Audits) - **see Case 1 below**
2. Multiple audit packages are created and one Audit Start is initiated between two Audit Ends (two Proof-of-Audits) - **see Case 2 below**
    - Sometimes, an E-SDC can initiate an Audit Start before submitting all the audit packages from that Audit. In that case, the system will wait for the last audit package from that Audit to arrive before it starts generating the Audit End command (Proof-of-Audit) - **see Case 2a below**
3. Multiple audit packages are created and multiple Audit Starts are initiated between two Audit Ends (two Proof-of-Audits) - **see Case 3 below**

Case 1 – Audit is performed after the creation of each audit package

This is the simplest case, where no additional audit packages are generated during the whole audit process, as follows:

1. Create an audit package
2. Initiate the Audit process by invoking the Start Audit APDU command
3. Receive a proof of audit and pass it to the End Audit APDU command
4. If EndAudit returns the value "true", you can safely delete the audit package(s)
5. If EndAudit returns the value "false", continue until a valid proof of audit is received
6. The period until the next Audit Start must be **at least 10 minutes (recommended from 30 minutes to a couple of hours)** after the previous Audit Start

The figure below illustrates the process:

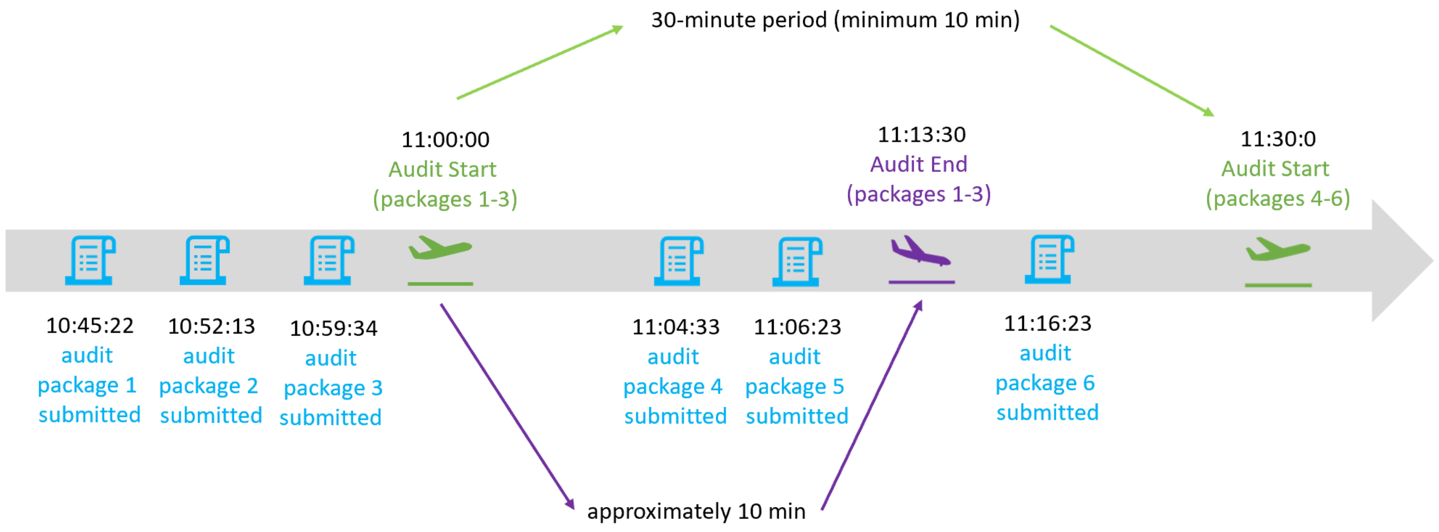


Case 2 – Audit is performed after multiple audit packages have been created

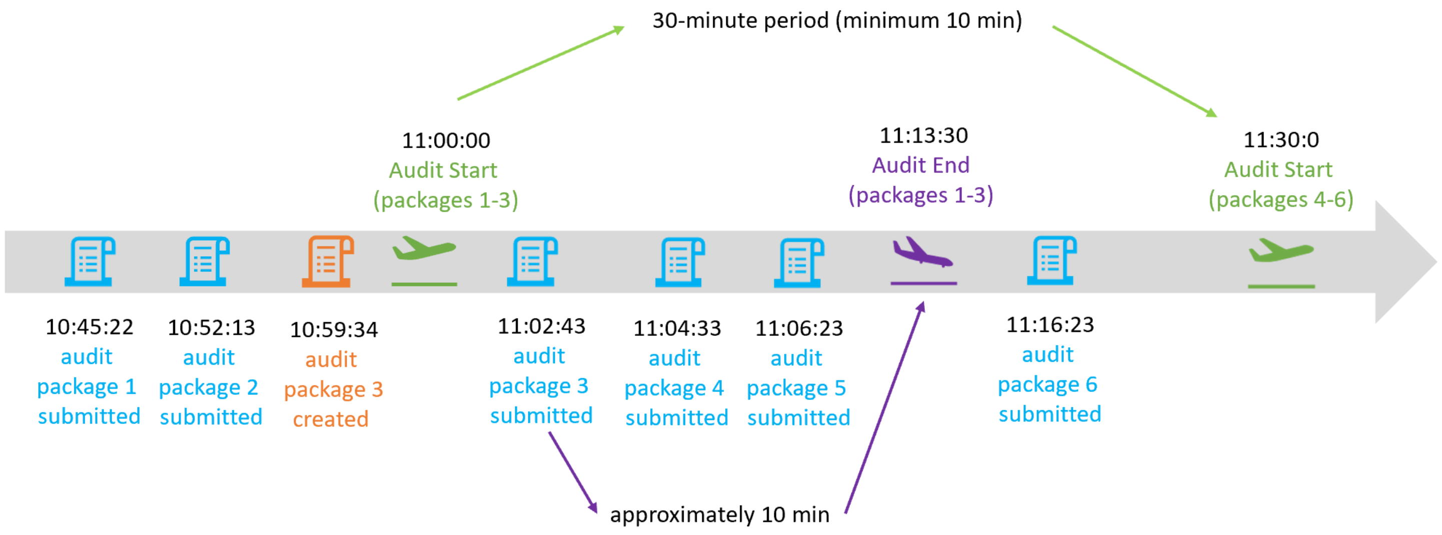
In this case, new packages can be created after an audit start:

1. Create audit packages 1-3 (as shown on the diagram)
2. Initiate the audit process by invoking the Start Audit APDU command
3. Continue to fiscalize invoices and create audit packages 4-6
4. Receive a proof of audit and pass it to the End Audit APDU command
5. If EndAudit APDU command returns value true you can delete remaining audit packages 1-3 because it is the last initial audit being invoked by E-SDC. Audit packages 4-6 are created after the call to BeginAudit APDU command so they are not audited in this cycle
6. If EndAudit APDU command returns value false, continue (return to point 1) until a valid proof of audit is received
7. The period until the next Audit Start must be **at least 10 minutes (recommended from 30 minutes to a couple of hours)** after the previous Audit Start

The figure below illustrates the process:


30-minute period (minimum 10 min)
11:00:00
Audit Start
(packages 1-3)
11:13:30
Audit End
(packages 1-3)
11:30:0
Audit Start
(packages 4-6)
10:45:22
audit package 1 submitted
10:52:13
audit package 2 submitted
10:59:34
audit package 3 submitted
11:04:33
audit package 4 submitted
11:06:23
audit package 5 submitted
11:16:23
audit package 6 submitted
approximately 10 min

Case 2a - Audit Start is initiated before all audit packages are submitted

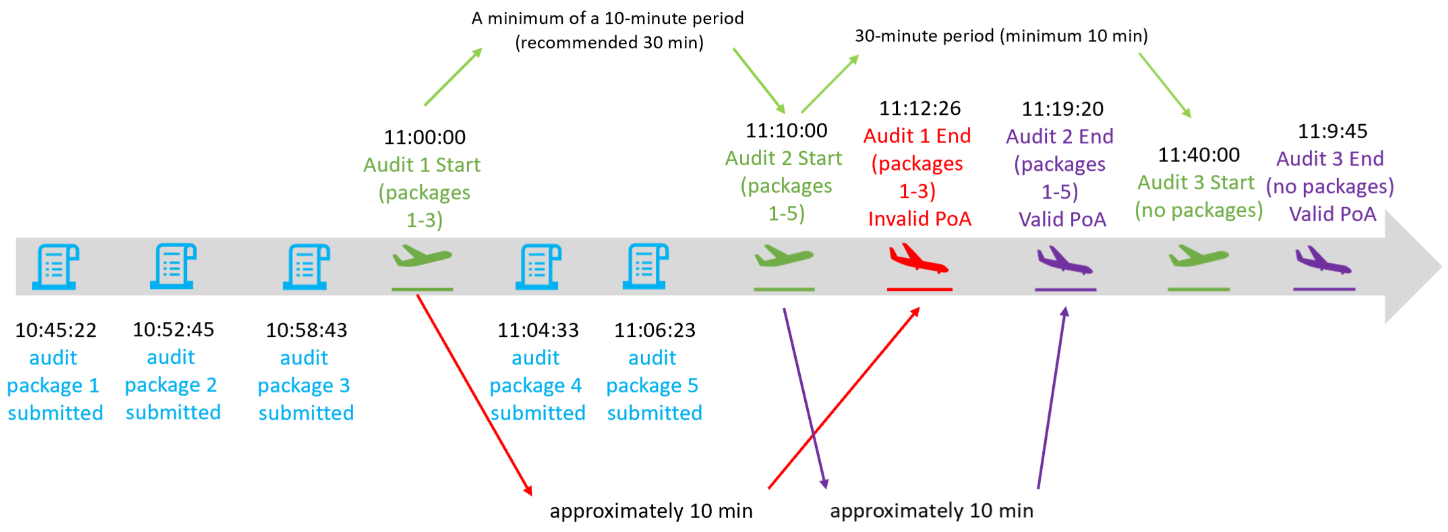


Case 3 – Audit is started multiple times before the Proof-of-Audit is generated

This case involves multiple audit starts:

1. Create Audit Packages 1-3
2. Initiate the Audit process by invoking the Start Audit APDU command
3. Continue to fiscalize invoices and create Audit Packages 4 and 5
4. Initiate another audit process by invoking the Start Audit APDU command – the previous audit is canceled
5. Receive the Proof-of-Audit and pass it to End Audit APDU command
6. If EndAudit returns value true you can delete remaining audit packages 1-5 because it is the last BeginAudit being invoked by E-SDC.
7. If EndAudit APDU command returns value false, continue until a valid Proof-of-Audit is received
8. A Proof-of-Audit generated for the first Audit Start (Audit 1 below) is not considered valid. Only the Proof-of-Audit which is generated for the last Audit Start (Audit 2 below) is considered valid and will be forwarded to the secure element.
9. The period until the next Audit Start must be **at least 10 minutes (recommended from 30 minutes to a couple of hours)** after the previous Audit Start

The figure below illustrates the process:



Notifications

Introduction

E-SDC device shall have an appropriate way to show the status of the device, information about the smart card and processes running on the E-SDC.

A cashier could get the device notifications by receiving an onscreen message, by observing the colors from the light-emitting diodes (LED) or any other similar component set for displaying visual notifications.

Required Notification

The following visual notifications shall be available to a cashier:

1. Smartcard is inserted but the E-SDC is not yet configured with the tax rates, verification URL or NTP service address. This is a common situation before initialization commands are executed by E-SDC;
2. Enter PIN Code for the Secure element – Smart card is inserted but E-SDC has not received the PIN Code from POS;
3. E-SDC is ready to sign an invoice;
4. Smart card is missing or unavailable;
5. Audit package transfer is in progress (Local audit on an SD card or USB flash drive, or an online audit);
6. Firmware update is in progress (if applicable);
7. Audit data storage is almost full;
8. Audit data storage is full;
9. Time for audit;
10. Commands in progress (currently running)

Switching Smart Cards During Operation

During normal operation, taxpayers/cashiers might switch the smart card they are using for issuing fiscal invoices.

In that case, the E-SDC must perform the following activities:

- **if the new smart card is for the same environment** - the E-SDC will first submit the unsubmitted invoices that were created with the previously used smart card
- **if the new smart card is for a different environment** - the E-SDC will keep the unsubmitted invoices (created with the previously used smart card) in its internal memory until they can be submitted (old smart card is returned)

For more information about different environments, see Identification of Environments and Important Endpoints.

E-SDC Logging

Introduction

E-SDC must keep a log about all required error events. It must log every error chronologically by local date and time (exact hour and minute).

E-SDC log must be available for easy export (download, USB flash drive...) and presented in a human-readable format.

Required Logging

The following error events must be logged:

- Any Invoice Request sent by POS that E-SDC failed to process
- Any APDU error returned by SE
- Any error returned by TaxCore
- Any error caused by internal E-SDC operations
- Any error during the E-SDC Initialization process (Enter PIN and Command processing).
- Any error during the Invoice Fiscalization process
- Any error during the Audit process (Local and Remote).
- Any error during the process of Date and Time Synchronization

The above errors are the minimum requirements, but E-SDC can also keep a log of other events.

Malfunctions and Non-serviceable Devices

Dump Audit Packages Kept on E-SDC when Secure element is damaged

If the Secure Element is damaged and its data cannot be restored from the card, but the E-SDC is operational, the tax authority system shall be able to dump data from the E-SDC device and upload the audit packages using the same application used to upload audit packages submitted by a taxpayer.

Protocols

This section describes Application Programming Interfaces (API) and protocols exposed by an E-SDC or used by an E-SDC to communicate with the other components (TaxCore.API, Secure element Applet, PKI Applet or SD Card/USB Flash Drive) required to fulfill its primary role – to safeguard a transaction and to transfer the audit packages to the tax authority's system.

Accredited POS systems can communicate with the E-SDC using the POS to SDC Protocol.

1. TaxCore.API
   TaxCore.API is a REST API exposed by a tax authority's system to E-SDC devices. It provides services used by the E-SDCs to submit Audit Packages, to notify TaxCore if the online status has been changed and to receive configuration commands.

2. Secure Element Applet API
   Communication with a Secure element Applet API is performed through standard APDU commands.

3. File Based Communication
   This section contains the description of the **File-based** communication with E-SDC.

TaxCore.API

TaxCore.API is a REST API exposed by a tax authority's system to E-SDC devices. It provides services used by the E-SDCs to submit Audit Packages, to notify TaxCore if the online status has been changed and to receive configuration commands.

An E-SDC is authenticated by TaxCore.API using a client digital certificate and an authentication Token.

Authentication

Introduction

Communication between a Client and TaxCore.API is carried out via the HTTPS protocol.

The Client is authenticated by TaxCore.API using either a client certificate or an authentication token obtained from TaxCore.API. To obtain an authentication token, a client certificate authentication has to be successfully conducted as the first step. For more information see Request Authentication Token.

Once token has been obtained HTTP request must contain *TaxCoreAuthenticationToken* key in the request HTTP headers with a valid authentication token as a value.

Digital Certificates and PIN Codes

The tax authority's system issues a Secure Element to a taxpayer as follows:

1. Taxpayer's digital certificate is stored in the Secure Element.
2. The Secure Element is stored on the smart card.
3. The PIN or password is generated and printed on the PIN mailer.
4. The Secure Element and PIN code are securely delivered to the taxpayer.

Digital Certificates for Testing Purpose

The tax authority will issue the requested number of test digital certificates to each accredited supplier and each accredited taxpayer.

Authentication Token

E-SDC uses an authentication token when calling the TaxCore API web services. Authentication token is obtained from TaxCore API by calling the service Request Authentication Token and providing a Taxpayer's digital certificate.

Role of the PKI Applet

PKI (public key infrastructure) Applet is installed along with the Secure Element Applet on the same Smart Card.

The role of the PKI applet is to support the secure communication and client certificate authentication with TaxCore.API using HTTPS protocol. The certificate used to establish a secure connection is stored on a smart card and it can be accessed from the PKI Applet using PKSC#11 API.

The certificate is loaded in the slot / token structure on the PKI Applet.

After the certificate is extracted from the smart card (in DER format) it can be used as a standard X.509 certificate for TLS/SSL and HTTPS protocols.

Valid PIN is required to read the certificate from PKI Applet using PKCS#11 API. Pin for PKI Applet is the same as the PIN for the Secure Element Applet.

Content

Required Drivers

Introduction

Smart Cards are programmed with PKI firmware according to GIDS (Generic Identity Device Specification) standard. Appropriate drivers shall be installed/programmed on an E-SDC in order to enable PKI Applet usage.

Windows OS Drivers

GIDS driver is an integral part of Windows OS since Windows 7 SP1, enabling the instant use of a smart card. No additional driver installation is required.

Linux OS Drivers

In order to use PKI Applet on Linux based OS, a pkcs11 driver from the OpenSC library is required. OpenSC libraries and tools are freely available on https://github.com/OpenSC.

In the following example, the installation of required drivers, libraries and tools on Debian / Ubuntu flavor of Linux OS with USB based card reader is shown. It is assumed that OpenSSL is used for TLS/SSL communication.

1. Install card reader driver

```
apt-get install libudev-dev
wget https://alioth.debian.org/frs/download.php/file/4126/pcsc-lite-x.y.z.tar.bz2
tar -xf pcsc-lite-x.y.z.tar.bz2
cd pcsc-lite-x.y.z
./configure
make
make install
```

```
aptitude install libusb-1.0-0-dev
wget https://alioth.debian.org/frs/download.php/file/4111/ccid-x.y.z.tar.bz2
tar -xf ccid-x.y.z.tar.bz2
cd ccid-x.y.z
./configure
make
make install
copy 92_pcscd_ccid.rules file from src directory to /etc/udev/rules.d/
# aptitude install libltdl-dev
wget http://ftp.de.debian.org/debian/pool/main/o/openct/openct_x.y.z.orig.tar.gz
tar -xf openct_x.y.z.orig.tar.gz
cd openct_x.y.z
./configure
# make
make all
```

2. Install OpenSSL development library

```
apt-get install libssl-dev
```

3. Install OpenSC package

```
wget http://cznic.dl.sourceforge.net/project/opensc/OpenSC/opensc-x.y.z/opensc-x.y.z.tar.gz
tar -xf opensc-x.y.z.tar.gz
cd opensc-x.y.z
./configure
make
make install
```

Run opensc-tool command from terminal

If message that libopensc.so.3 cannot be loaded find it with

```
find / -name "libopensc.so"
```

Copy found library to /usr/lib

4. Install libp11 library

```
apt-get install libp11-2
```

5. Install engine_pkcs11 library

- Download source code from https://github.com/OpenSC/engine_pkcs11/releases/
- Build and install library according to instructions found project page

After the above steps are executed, the certificate shall be accessible from the appropriate slot/token using a PKCS11 family of functions from the lipb11 library. ENGINE family of functions can be used to load the pkcs11 engine in the OpenSSL.

Other Platforms and Operating Systems

Please contact OpenSC community (https://github.com/OpenSC) for further information.

SDC

Introduction

SDC Section of TaxCore.API is used by any SDCs to establish connection with backend.

Activities

- Get initialization and configuration information required for normal operation
- Submit prepared audit packages
- Notify backend of online/offline status
- Pull pending commands from backend
- Notify backend of command execution results

Content

1. Request Authentication Token
   When requesting the authentication token, a Client authenticates itself with a valid Digital Certificate (stored in the PKI applet). If the token is successfully created it is returned to the Client as a string. In order to receive an authentication token, each client must establish a secure connection to "/api/v3/sdc/token" endpoint on TaxCore.API and authenticate using client digital certificate.

2. Get Initialization Commands
   For each new smart card issued by a tax authority, a set of commands is generated, which contain information necessary for invoice signing (Tax Rates, Verification URL, NTP server etc.).

3. Notify Online Status
   If an E-SDC is online, it shall periodically (once every 1 – 5 minutes) invoke the "Notify Online Status" operation on TaxCore.API.

4. Notify Command Processed
   After an E-SDC processes commands received from TaxCore.API, it will report the results of execution to TaxCore.API.

5.

Submit Audit Package
After the invoice audit package is created (explained in the section Creating an Audit Package), it shall be transferred to TaxCore.API the next time an Internet connection is available.

6.

Submit Audit Request Payload ARP
E-SDC invokes the Start Audit APDU command and receives 260 bytes of data that represent the Audit Request Payload (ARP). ARP has to be converted to the string using Base64 encoding.

Request Authentication Token

Introduction

When requesting the authentication token, a Client authenticates itself with a valid Digital Certificate (stored in the PKI applet). If the token is successfully created it is returned to the Client as a string. In order to receive an authentication token, each client must establish a secure connection to “/api/v3/sdc/token” endpoint on TaxCore.API and authenticate using client digital certificate.

A request is composed as follows:

1. Create HTTPS GET request object
2. Add HTTP headers "Accept: application/json" and "Content-Type: application/json"
3. Read certificate from the PKI Applet
4. Use the certificate from the PKI Applet to establish SSL/TLS connection
5. Send a request to "/api/v3/sdc/token" operation on TaxCore.API web service.
6. Read the response as JSON structure defined below

Endpoints

Endpoint	Example
<TaxCore_API_URL_obtained_from_certificate_as_e here>/api/v3/sdc/token	`https://api.sandbox.suf.poreskaupravars.c`

NOTE:
Development and production environments, as well the environments in different countries, have different URLs. For this reason, URLs and names in your documentation, code and UI should not be hardcoded but configurable or extracted from a digital certificate.

Method

`GET`

Header

Add the following HTTP headers to each request:

- `Accept`: `application/json`

Authentication

The certificate-based authentication is used only to request a token. Request for the authentication token is periodically invoked to obtain a new token and to verify the date and time.

For authentication details please refer to Authentication

Request

N/A

Response

Field	Type	Description
token	string	The Token is valid for 8 hours by default. A Client uses the current token when calling all other services exposed by TaxCore.API.
expiresAt	string	Date and time of token expiration - in UTC time. When a token expires a Client must request a new token. If the Client requests a new token while the current token is still valid, TaxCore will return the current token.

Example

```
{
  "token": "245ebd69-1438-4dc3-a65b-18f1a527f093",
  "expiresAt": "2020-12-23 15:18:33Z"
}
```

Get Initialization Commands

Introduction

For each new smart card issued by a tax authority, a set of commands is generated, which contain information necessary for invoice signing (Tax Rates, Verification URL, NTP server etc.).

Commands can be downloaded using one of the following channels:

- By invoking TaxCore.API service Notify Online Status (typically by E-SDC)
- By invoking TaxCore.API service Submit Audit Package (typically by E-SDC)
- By invoking TaxCore.API service Get Initialization Commands (typically by E-SDC)
- By using the Taxpayer Administration Portal

Once the commands are processed, E-SDC reports the execution status to TaxCore.API as explained in section Notify Command Processed.

E-SDC can explicitly require initialization commands, by invoking TaxCore.API service *Get Initialization Commands*.

Initialization commands include:

- *Configure Time Server URL Command*
- *Set Tax Rates Command*
- *Update Verification URL Command*
- *Update TaxCore Configuration*

To get initialization commands compose HTTPS GET request as follows:

1. Add headers "Accept: application/json", "Content-Type: application/json" and header that contains an authentication token
2. Submit GET request to https://<taxcore_api_url>/api/v3/sdc/commands

Endpoints

Endpoint	Example
<TaxCore_API_URL_obtained_from_certificate_as_e here>/api/v3/sdc/commands	`https://api.sandbox.suf.poreskaupravars.c`

NOTE:
Development and production environments, as well the environments in different countries, have different URLs. For this reason, URLs and names in your documentation, code and UI should not be hardcoded but configurable or extracted from a digital certificate.

Method

```
GET
```

Header

Add the following HTTP headers to each request

- `TaxCoreAuthenticationToken: <token-value-returned-from-Request-Authentication-Token-method>`
- `Accept: application/json`

Authentication

For authentication details please refer to Authentication

Request

N/A

Response

The response contains a list of commands that must be executed by E-SDC, as described in section Commands.

Notify Online Status

Introduction

If an E-SDC is online, it shall periodically (once every 1 – 5 minutes) invoke the "Notify Online Status" operation on TaxCore.API.

Compose HTTPS PUT request as follows:

1. Add headers "Accept: application/json", "Content-Type: application/json" and header that contains authentication token
2. Add a string "true" or "false" to the body of the request (depending on whether the E-SDC is online or going offline)
3. Submit PUT request to https://<taxcore_api_url>/api/v3/sdc/status

After the request is sent, TaxCore.API shall return a response with a JSON formatted string containing a list of commands, as described in section Commands, that an E-SDC shall execute (new tax rates, verification URL, NTP URL or public key used for encryption). The Command list can be empty.

Endpoints

Endpoint	Example
`https://api.sandbox.suf.poreskaupravars.c`	`https://api.sandbox.taxcore.online/api/`

Method

`PUT`

Header

Add the following HTTP headers to each request

- `TaxCoreAuthenticationToken: <token-value-returned-from-Request-Authentication-Token-method>`
- `Accept: application/json`
- `Content-Type: application/json`

Authentication

For authentication details please refer to Authentication

Request

true

The list of commands will be obtained only if the string "true" is submitted in the request

Response

The response contains a list of commands that must be executed by E-SDC, as described in section Commands.

Notify Command Processed

Introduction

After an E-SDC processes commands received from TaxCore.API, it will report the results of execution to TaxCore.API.

1. Add headers "Accept: application/json", "Content-Type: application/json" and header that contains authentication token
2. Add a string “true” or "false" to the body of the request (depending on whether the E-SDC successfully processed commands)
3. Submit PUT request to https://<taxcore_api_url>/api/v3/sdc/commands/{commandId}

Endpoints

Endpoint	Example
<TaxCore_API_URL_obtained_from_certificate_as_e: here>/api/v3/sdc/commands/{commandId}	`https://api.sandbox.suf.poreskaupravars.c 205A-4CBF-AD0C-6617D42AE466`

NOTE:
Development and production environments, as well the environments in different countries, have different URLs. For this reason, URLs and names in your documentation, code and UI should not be hardcoded but configurable or extracted from a digital certificate.

Method

```
PUT
```

Header

Add the following HTTP headers to each request

- `TaxCoreAuthenticationToken: <token-value-returned-from-Request-Authentication-Token-method>`
- `Accept: application/json`
- `Content-Type: application/json`

Authentication

For authentication details please refer to Authentication

Request

true

Example

https://api.sandbox.taxcore.online/api/v3/sdc/commands/CC63C53D-205A-4CBF-AD0C-6617D42AE466

Response

HTTP 200 OK

Submit Audit Package

After the invoice audit package is created (explained in the section Creating an Audit Package), it shall be transferred to TaxCore.API the next time an Internet connection is available.

Compose HTTPS POST request as follows:

1. Add headers "Accept: application/json", "Content-Type: application/json" and header that contains an authentication token
2. Add an Audit Package as a JSON message to the body of the HTTP POST request
3. Submit POST request to https://<taxcore_api_url>/api/v3/sdc/audit

After the request is sent, TaxCore.API responds with a JSON formatted text containing a status of operation and a list of commands that an E-SDC shall execute.

NOTE:
Values for `sdcDateTime` and all other fields (see Create Invoice) must match the values submitted to the Secure Element for digital signing (see *Sign Invoice* in Fiscalization), as well as the values in the verification URL (see Create Verification URL).

NOTE:
In case the field `Request.Item.Name` exceeds the defined maximum length (see Create Invoice), its value is truncated.

NOTE:
In case the field `Request.DateAndTimeOfIssue` is out of range (see Create Invoice), its value is replaced with a null value.

Endpoints

Endpoint	Example
<TaxCore_API_URL_obtained_from_certificate_as_e: here>/api/v3/sdc/audit	`https://api.sandbox.suf.poreskaupravars.c`

NOTE:
Development and production environments, as well the environments in different countries, have different URLs. For this reason, URLs and names in your documentation, code and UI should not be hard-coded but configurable or extracted from a digital certificate.

Method

`POST`

Header

Add the following HTTP headers to each request

- `TaxCoreAuthenticationToken: <token-value-returned-from-Request-Authentication-Token-method>`
- `Accept: application/json`
- `Content-Type: application/json`

Authentication

For authentication details, please refer to Authentication.

Request

Request that must be generated and sent by SDC is described in this section Format of the Audit Package.

Example

```
{
  "key": "VGhpcyBJcyBLZXkK...",
  "iv": "VGhpcyBJcyBJVgo...",
  "payload": "VGhpcyBJcyBQYXlsb2FkCg...",
  "invoicenumber": "QWERASDF-QWERASDF-54515"
}
```

Response

The response contains a list of commands that must be executed by E-SDC, as described in section Commands.

```
AuditDataStatus {
status (integer, optional) = ['0', '1', '2', '3', '4', '5', '6']integerEnum:0, 1, 2, 3, 4, 5,
commands (Array[Command], optional)
}]
```

Data Fields

status - returned after TaxCore.API unpacks and verifies audit packages. If all verifications are successful, the status should have the value **4**. Other values can help E-SDC developers rectify problems with audit packages. Their meanings are the following:

- **0** - Invalid audit package - TaxCore API failed to decrypt the received audit package. The possible reasons are: the received file is corrupt, has no content, the file is encrypted using the wrong TaxCore public key, etc.
- **1** - Invoice cannot be stored - the received invoice is probably valid, but due to the internal server error TaxCore.API is unable to store it
- **72** - E-SDC sent the wrong TIN or SignedBy or RequestedBy fields when signing this invoice
- **3** - Invoice internal data was encrypted in a wrong way
- **4** - Invoice is verified
- **5** - Taxes on this invoice were calculated using a tax rates group that does not exist or is obsolete
- **6** - At the moment of invoice signing, the certificate on the secure element was already revoked
- **8** - At the moment of invoice signing, the certificate on the secure element was not officially issued
- **23** - Information about the E-SDC's Manufacturer Registration Code is missing
- **24** - Information about the E-SDC's Manufacturer Registration Code is in a wrong format
- **67** - SDC time is out of the allowed range

- **68** - Reference time is out of the allowed range
- **69** - Invoice contains TaxItem for a label that does not exist in the tax rate group named in `Result.TaxGroupRevision`
- **70** - Invoice contains TaxCounters (in internal data) for CategoryOrderId which does not exist in the tax rate group named in `Result.TaxGroupRevision`, i.e. E-SDC sent a non-existing CategoryOrderId to the secure element
- **71** - The invoice was submitted without information about the payment
- **73** - Invoice contains SDC time (`result.sdcDateTime`) that is in the future compared to the time of invoice arrival to TaxCore.API. The gap is greater than the allowed tolerance period.
- **74** - Invoice contains Reference time (`request.referentDocumentDT`) that is in the future compared to the invoice's SDC time (`result.sdcDateTime`). The gap is greater than the allowed tolerance period.
- **75** - Invoice contains SDC time (`result.sdcDateTime`) that is in the past compared to the time of invoice arrival to TaxCore.API. The gap is greater than the allowed tolerance period.
- **76** - Invoice contains Reference time (`request.referentDocumentDT`) that is in the past compared to the invoice's SDC time (`result.sdcDateTime`). The gap is greater than the allowed tolerance period.
- **78** - Invoice does not contain a Verification URL
- **commands** - contains a list of commands that E-SDC should execute, as described in section Commands.

NOTE:
If status 4 (*Invoice is verified*) is received from TaxCore.API, that audit package should immediately be deleted from the E-SDC's local storage (see Creating an Audit Package). If any other status is received, the audit package must not be deleted.

NOTE:
The E-SDC should **try to resubmit** an audit package to TaxCore.API. **only** if it receives status 1 (*Invoice cannot be stored*) from the TaxCore.API. If any other status is received, the audit package should not be resubmitted.

Example

```
{
  "status": 0,
  "commands": [
    {
      "commandId": "3930CEEF-F637-444D-8295-F629D6E482D3",
      "type": 1,
      "payload": "0.europe.pool.ntp.org",
      "uid": "ABCD1234"
    }
  ]
}
```

Submit Audit Request Payload - ARP

E-SDC invokes the Start Audit APDU command and receives 260 bytes of data that represent the Audit Request Payload (ARP). ARP has to be converted to the string using Base64 encoding.

E-SDC invokes the Amount Status APDU command and receives the current sum and limit for the secure element.

These 3 values are submitted to endpoint https://<taxcore_api_url>/api/v3/sdc/audit-proof as an Audit-Proof Request structure in the body of the HTTP request.

Compose HTTPS POST request as follows:

1. Add headers "Accept: application/json", "Content-Type: application/json" and header that contains authentication token
2. Create a request structure as per the below model and add it to the body of the HTTP POST request
3. Submit POST Request to https://<taxcore_api_url>/api/v3/sdc/audit-proof

Endpoints

Endpoint	Example
<TaxCore_API_URL_obtained_from_certificate_as_e: here>/api/v3/sdc/audit-proof	`https://api.sandbox.suf.poreskaupravars.c proof`

NOTE:
Development and production environments, as well the environments in different countries, have different URLs. For this reason, URLs and names in your documentation, code and UI should not be hardcoded but configurable or extracted from a digital certificate.

Method

```
POST
```

Header

Add the following HTTP headers to each request

- `TaxCoreAuthenticationToken: <token-value-returned-from-Request-Authentication-Token-method>`
- `Accept: application/json`
- `Content-Type: application/json`

Authentication

For authentication details please refer to Authentication

Request

JSON structure as defined in section Format of the Audit-Proof Request

Model

```
ProofOfAuditRequest {
auditRequestPayload (string),

sum (integer) 64bit unsigned,

limit (integer) 64bit unsigned
}
```

Example

```
{
  "auditRequestPayload": "d4A/iLtwmDYeZyacm/nDlCF...",
  "sum": 11034,
  "limit": 100000
}
```

Response

```
HTTP 200 OK
```

Secure Element Applet API

Communication with a Secure element Applet API is performed through standard APDU commands.

For a detailed description of APDU communication, APDU commands data structure and particular bytes meaning, please refer to ISO/IEC 7816-4 standard.

Commands are grouped into three categories based on the type of usage:

1

General
2. Fiscalization
3. Audit

Important Notes

1. All APDU commands are sent to the Smart Card using T1 communication protocol
2. All amounts or counter values are submitted to/received from the Secure element using Big-endian. Big-endian is an order in which the "big end" (most significant value in the sequence) is stored first (at the lowest storage address)
3. P1 and P2 values considered in the request processing when,
    1. Select Applet Command
    2. force using CRC for Data in APDU transimission
4. PIN is sent in ASCII hex format from SE applet version 3.2.2.
5. CRC is available from SE applet version 3.2.5, and it is optional to use.

Content

1. General Commands
   Secure Element Applet is installed as a non-default applet on a smart card. Before any APDU command is invoked, the applet is selected using the standard Select command.

2. Fiscalization
   PIN verification is a method that “unlocks” a card for invoice signing and other operations protected by PIN code. Depending on the SE applet version, PIN is sent in decimal or hex format with ASCII encoding, and it is sent as an array of byte digits.

3. Audit
   Returns 259 bytes data structure represents public card key (256 bytes modulus and 3 bytes exponent). This key is used to encrypt Audit packages.

4. Secure Element Specific APDU Error Codes
   This table contains the expected error codes and descriptions that a caller may encounter while working with the Secure Element Applet.

General Commands

Secure Element Applet is installed as a non-default applet on a smart card. Before any APDU command is invoked, the applet is selected using the standard Select command.

NOTE:
The availability of specific commands, as well as their content, depends on the secure element (SE) version. You can use the *Get Secure Element Version* command (see below) to check the version of the SE you are using.

Select Applet

As previously mentioned, the Smart Card has two applets installed. This command selects the Secure Element Applet and routes subsequent APDU commands to it.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expecte Lengt (Le)
>= 2.0.0	`Case3Sho`	`0x00`	`0xA4`	`0x040`	`0x10`	`0xA000000748`	`0x00`

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0	none	`0x9000`

Example:

Request: `00A4040010A000000748464A492D546178436F726500`

Response: `9000`

Get Secure Element Version

This command returns the version information about the current Api version. The response contains 12 bytes, where each 4 bytes represent unsigned integer of one version segment, making total of 3 version segments: major, minor and patch.

APDU Request

SE CAP Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)

>= 2.0.0	`Case2Sho`	`0x88`	`0x08`	`0x000(`	none	none	`0x00`

APDU Response

SE CAP Version	Response Data	SW1SW2
>= 2.0.0	*12 bytes*	`0x9000`

Example 1:

Request: `8808040000`

Response: `000000020000000000000000 9000`

Example 2:

Request: `8808000000`

Response: `000000030000000100000001 9000`

Example 3:

Request: `8808000000`

Response: `000000030000000200000005 9000`

Forward Secure Element Directive

This command is used by E-SDC to forward instructions received from TaxCore.Api to Secure Element Applet via Secure Element APDU Command.

If APDU Command status (SW1SW2) is OK (`0x9000`), consider forward instructions operation is completed.

NOTE:
From the SE version 3.2.5, optionally, CRC can be calculated and used for data verification. If CRC is not used, the command is the same as in the previous applet version.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0	`Case3Ext`	`0x88`	`0x40`	`0x040(`	`0x000200`	*512 bytes*	none

(no CRC)						*received from TaxCore*	
>= 3.2.5 (with CRC)	`Case3Ext`	`0x88`	`0x40`	`0x0102`	`0x000204`	*512 bytes received from TaxCore + 4 bytes for CRC*	none

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0 (no CRC)	none	`0x9000`
>= 3.2.5 (with CRC)	none	`0x9000`

Example 1 (without CRC):

Command Data:
`5DBFC9CD04AF9DC76C50FA3FF54D32D1910B0D2E1EC5AF97EAE3E71A7423CCE066D6E264255838C1DBAD`

Request:
`884004000002005DBFC9CD04AF9DC76C50FA3FF54D32D1910B0D2E1EC5AF97EAE3E71A7423CCE066D6E2`

Response: `9000`

Example 2 (with CRC):

Command Data witout CRC:
`5DBFC9CD04AF9DC76C50FA3FF54D32D1910B0D2E1EC5AF97EAE3E71A7423CCE066D6E264255838C1DBAD`

Command Data CRC: `F50CFF4B`

Command Data:
`5DBFC9CD04AF9DC76C50FA3FF54D32D1910B0D2E1EC5AF97EAE3E71A7423CCE066D6E264255838C1DBAD`

Request:
`884004000002045DBFC9CD04AF9DC76C50FA3FF54D32D1910B0D2E1EC5AF97EAE3E71A7423CCE066D6E2`

Response: `9000`

Export Certificate

This command exports the taxpayer certificate in a DER format. This certificate contains location data that is present on the textual representation of an invoice.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0	`Case2Ext`	`0x88`	`0x04`	`0x040`	none	none	`0x000000`

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0	*raw bytes random length*	`0x9000`

Example:

Request: `88040400000000`

Response: *raw bytes of x509 certificate public key* + `9000`

Get Last Signed Invoice

This command returns information about the last singed invoice. The structure of the data recived is the same as the response is in the Sign Invoice command.

NOTE:
From the SE version 3.2.5, optionally, CRC can be calculated and used for data verification. If CRC is not used, the command is the same as in the previous applet version.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 3.1.1 (no CRC)	`Case2Ext`	`0x88`	`0x15`	`0x040`	none	none	`0x000000`
>= 3.2.5 (with CRC)	`Case2Ext`	`0x88`	`0x15`	`0x010`	none	none	`0x000000`

APDU Response

SE Version	Response Data	SW1SW2
>= 3.1.1 (no CRC)	*577 or 833 bytes*	`0x9000`
>= 3.2.5 (with CRC)	*581 or 837 bytes*	`0x9000`

Example 1 (without CRC):

Request: `88150400000000`

Response: *577 or 833 bytes* + `9000`

Response Data

Start (byte)	Length (bytes)	Field	Description
0	8	Date/time	Same as data sent from E-SDC to SE
8	20	Taxpayer ID	Same as data sent from E-SDC to SE
28	20	Buyer ID	Same as data sent from E-SDC to SE
48	1	Invoice type	Same as data sent from E-SDC to SE
49	1	Transaction type	Same as data sent from E-SDC to SE
50	7	Invoice amount	Same as data sent from E-SDC to SE
57	4	Sale or refund counter value	Depends on request's Tax type field
61	4	Total counter value (sale+refund)	Unsigned int 32bit big endian,
65	256 or 512	Encrypted Internal Data	Encrypted Internal Data length depends on the number of available tax rates programmed during personalization. It may be 256 or 512 bytes long.
321 or 577	256	Digital signature	

Example 2 (with CRC):

Request: `8815010200`

Response: *581 or 837* + `9000`

Response Data

Start (byte)	Length (bytes)	Field	Description
0	8	Date/time	Same as data sent from E-SDC to SE
8	20	Taxpayer ID	Same as data sent from E-SDC to SE
28	20	Buyer ID	Same as data sent from E-SDC to SE
48	1	Invoice type	Same as data sent from E-SDC to SE
49	1	Transaction type	Same as data sent from E-SDC to SE
50	7	Invoice amount	Same as data sent from E-SDC to SE
57	4	Sale or refund counter value	Depends on request's Tax type field
61	4	Total counter value (sale+refund)	Unsigned int 32bit big endian,
65	256 or 512	Encrypted Internal Data	Encrypted Internal Data length depends on the number of available tax rates programmed during personalization. It may be 256 or 512 bytes long.
321 or 577	256	Digital signature	
577 or 833	4	CRC	CRC is calculated from 0 to 577 or 833 bytes.

Get PIN tries left from SE Applet

This command returns how many PIN tries are left before the card is locked

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 3.1.1	`Case2Sho`	`0x00`	`0x16`	`0x040(`	none	none	`0x00`

APDU Response

SE Version	Response Data	SW1SW2
>= 3.1.1	05 if 5 tries are left, 00 if the card is blocked	`0x9000`

Example:

Request: `8816040000`

Response: `05 9000`

Fiscalization

PIN Verify

PIN verification is a method that “unlocks” a card for invoice signing and other operations protected by PIN code. Depending on the SE applet version, PIN is sent in decimal or hex format with ASCII encoding, and it is sent as an array of byte digits.

For example, PIN 1234 can be represented in the following formats:

- decimal format - PIN is represented as 0x01, 0x02, 0x03, 0x04.
- ASCII hex format - PIN is represented as 0x31, 0x32, 0x33, 0x34.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
2.0.0 ￿ SE version <	`Case3Sho`	`0x88`	`0x11`	`0x000(`	`0x04`	*4 bytes where*	none

3.2.2						*each represents one PIN digit in decimal format*	
>= 3.2.2	`Case3Sho`	`0x88`	`0x11`	`0x000(`	`0x04`	*4 bytes where each represents one PIN digit in ASCII hex format*	none

Example:

This is an example for PIN 1234.

SE Version	Command Data	Request	Response (correct PIN)	Error response (wrong PIN)
2.0.0 □ SE version < 3.2.2	`01020304`	`88110000040102030`	`9000`	`6302`
>= 3.2.2	`31323334`	`88110000043132333`	`9000`	`6302`

Sign Invoice

Signs invoice and returns fiscalization data for a submitted invoice.

NOTE:
From the SE version 3.2.5, optionally, CRC can be calculated and used for data verification. If CRC is not used, the command is the same as in the previous applet version.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0 (no CRC)	`Case4Ext`	`0x88`	`0x13`	`0x040(`	*3 byte Command Data byte*	*Command Data byte array*	`0x0000`

					array length		
>= 3.2.5 (with CRC)	`Case4Ext`	`0x88`	`0x13`	`0x0102`	*3 byte Command Data byte array length*	*Command Data byte array + 4 bytes for CRC*	`0x0000`

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0 (no CRC)	*byte array*	`0x9000`
>= 3.2.5 (with CRC)	*byte array + 4 byte CRC*	`0x9000`

Data structure without CRC:

Command data:

Start (byte)	Length (byte)	Field	Description
0	8	Date/time	E-SDC timestamp UTC time in Unix Timestamp. Example: 1495018011910 is 2017-05-17T10:46:51.910Z
8	20	Taxpayer ID	Hex encoded byte array, leading bytes filled with 0x00. Taxpayer ID value can consist only of ascii printable characters. **Zeros can be added only on the left side**. MSB are sent first Example: Taxpayer ID = 928615467, Byte array = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x39, 0x32, 0x38, 0x36, 0x31, 0x35, 0x34, 0x36, 0x37} (byte 0x37 is sent last to SE)
28	20	Buyer ID	If unknown, leave zeroes. Formatting is the same as for Taxpayer ID
48	1	Invoice type	Values 0, 1, 2, 3, 4 as explained in section Create Invoice.
49	1	Transaction Type	Sale=0, Refund=1

50	7	Invoice amount	Sale or refund total amount (including taxes) - depends on applied tax types
57	1	Number of tax categories	Defines the number of tax categories which appear on the invoice (value between 0 and 26). The following data structure **Tax Categories** must be repeated exactly this number of times.
58	8	Tax Category (1)	The first Tax Category (mandatory if **Number of tax categories > 0**)
66	8	Tax Category (2)	The second Tax Category (mandatory if **Number of tax categories > 1**)
74	...	Tax Category (n)	

Tax Categories:

Start (byte)	**Length (byte)**	**Field**	**Description**
58	[1]	[Tax category ID]	The first tax category's OrderID, as explained in Tax Rates section (mandatory if **Number of tax categories** > 0)
59	[7]	[Tax category amount]	The first total tax amount for the category specified in preceding field **Tax category ID** (mandatory if **Number of tax categories** > 0)
66	[1]	[Tax category ID]	The next tax category's OrderID (mandatory if **Number of tax categories** > 1)
67	[7]	[Tax category amount]	The next total tax amount for the category specified in preceding field **Tax category ID** (mandatory if **Number of tax categories** > 1)

Response data:

Start (byte)	**Length (bytes)**	**Field**	**Description**
0	8	Date/time	Same as data sent from E-SDC to SE
8	20	Taxpayer ID	Same as data sent from E-SDC to SE
28	20	Buyer ID	Same as data sent from E-SDC to SE

48	1	Invoice type	Same as data sent from E-SDC to SE
49	1	Transaction type	Same as data sent from E-SDC to SE
50	7	Invoice amount	Same as data sent from E-SDC to SE
57	4	Sale or refund counter value	Depends on request's Tax type field
61	4	Total counter value (sale+refund)	unsigned int 32bit big endian,
65	256 or 512	Encrypted Internal Data	Encrypted Internal Data length depends on the number of available tax rates programmed during personalization. It may be 256 or 512 bytes long.
321 or 577	256	Digital signature	

Example without CRC:

Command Data:
`000001 7B2D198AC40000000000000000 50432D313030303030303031 0000000000000000000000000000`

Request:
`8813040000009A0000017BE9B01AB4000000000000000050432D31303030303030303100000000000000`

Response: *byte array* + `9000`

Data structure with CRC:

Command data:

Start (byte)	Length (byte)	Field	Description
0	8	Date/time	E-SDC timestamp UTC time in Unix Timestamp. Example: 1495018011910 is 2017-05-17T10:46:51.910Z
8	20	Taxpayer ID	Hex encoded byte array, leading bytes filled with 0x00. Taxpayer ID value can consist only of ascii printable characters. **Zeros can be added only on the left side**. MSB are sent first Example: Taxpayer ID = 928615467, Byte array = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x39, 0x32, 0x38, 0x36, 0x31, 0x35, 0x34, 0x36, 0x37} (byte 0x37 is sent last to SE)

28	20	Buyer ID	If unknown, leave zeroes. Formatting is the same as for Taxpayer ID
48	1	Invoice type	Values 0, 1, 2, 3, 4 as explained in section Create Invoice.
49	1	Transaction Type	Sale=0, Refund=1
50	7	Invoice amount	Sale or refund total amount (including taxes) - depends on applied tax types
57	1	Number of tax categories	Defines the number of tax categories which appear on the invoice (value between 0 and 26). The following data structure **Tax Categories** must be repeated exactly this number of times.
58	8	Tax Category (1)	The first Tax Category (mandatory if **Number of tax categories > 0**)
66	8	Tax Category (2)	The second Tax Category (mandatory if **Number of tax categories > 1**)
74	...	Tax Category (n)	
...	4	CRC	CRC is calculated from 0 to 74 bytes (or to last byte if data).

Tax Categories:

Start (byte)	**Length (byte)**	**Field**	**Description**
58	[1]	[Tax category ID]	The first tax category’s OrderID, as explained in Tax Rates section (mandatory if **Number of tax categories** > 0)
59	[7]	[Tax category amount]	The first total tax amount for the category specified in preceding field **Tax category ID** (mandatory if **Number of tax categories** > 0)
66	[1]	[Tax category ID]	The next tax category’s OrderID (mandatory if **Number of tax categories** > 1)
67	[7]	[Tax category amount]	The next total tax amount for the category specified in preceding field **Tax category ID** (mandatory if **Number of tax categories** > 1)

Response data:

Start (byte)	Length (bytes)	Field	Description
0	8	Date/time	Same as data sent from E-SDC to SE
8	20	Taxpayer ID	Same as data sent from E-SDC to SE
28	20	Buyer ID	Same as data sent from E-SDC to SE
48	1	Invoice type	Same as data sent from E-SDC to SE
49	1	Transaction type	Same as data sent from E-SDC to SE
50	7	Invoice amount	Same as data sent from E-SDC to SE
57	4	Sale or refund counter value	Depends on request's Tax type field
61	4	Total counter value (sale+refund)	unsigned int 32bit big endian,
65	256 or 512	Encrypted Internal Data	Encrypted Internal Data length depends on the number of available tax rates programmed during personalization. It may be 256 or 512 bytes long.
321 or 577	256	Digital signature	
577 or 833	4	CRC	CRC is calculated from 0 to 577 or 833 bytes.

Example with CRC:

Command Data:
`0000017B2D198AC4000000000000000050432D31303030303030303100000000000000000000000000000000`

Command Data CRC: `90F2BC39`

Request:
`88130102E0000017BE9B01AB4000000000000000050432D313030303030303031000000000000000000`

Response: *byte array invoice + 4 byte CRC +* `9000`

Amount Status

Returns 14-bytes-long data structure (7 bytes for sum SALE and REFUND, and 7 bytes for Limit Amount)

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0	`Case2Sho`	`0x88`	`0x14`	`0x040(`	none	none	`0x00`

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0	*14 byte array*	`0x9000`

Example:

Request: `8814040000`

Response: `0000724AA18328038D7EA4C68000 9000` (SALE+REFUND=490878370600 , Limit Amount=1000000000000000)

Audit

Export TaxCore Public Key

Returns 259 bytes data structure represents public card key (256 bytes modulus and 3 bytes exponent). This key is used to encrypt Audit packages.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0	`Case2Ext`	`0x88`	`0x07`	`0x040(`	none	none	`0x000000`

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0	*259 bytes data*	`0x0900`

Example:

Request: `88070400000000`

Response: *256 bytes modulus + 3 bytes exponent +* `9000`

Export Audit Data

Exports encrypted audit data.

NOTE:
From the SE version 3.2.5, optionally, CRC can be calculated and used for data verification. If CRC is not used, the command is the same as in the previous applet version.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0 (no CRC)	`Case2Ext`	`0x88`	`0x12`	`0x040`	none	none	`0x000000`
>= 3.2.5 (with CRC)	`Case2Ext`	`0x88`	`0x12`	`0x010`	none	none	`0x000000`

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0 (no CRC)	*565 or 821 bytes data*	`0x9000`
>= 3.2.5 (with CRC)	*569 or 825 bytes data*	`0x9000`

NOTE:
Depending on the Internal Data, the total length of the structure is 565 or 821 bytes. For versions **3.2.5 or later** if CRC is used, the total lenght can be 569 or 825 if CRC is added.

Exported audit data has the following structure, without CRC:

Offset	Length	Data	Note
0	4	TaxCore Key Version	
4	256	Crypted Internal Data	The length of Crypted Internal Data can be 256 or 512 bytes
260 or 516	20	Taxpayer Identification Number (TIN)	
280 or 536	20	Buyer ID	
300 or 556	1	Invoice type	
301 or 557	1	Transaction type	
302 or 558	7	Invoice amount	
309 or 565	256	Digital signature of the above structure	

Exported audit data has the following structure, with CRC:

Offset	Length	Data	Note
0	4	TaxCore Key Version	
4	256	Crypted Internal Data	The length of Crypted Internal Data can be 256 or 512 bytes
260 or 516	20	Taxpayer Identification Number (TIN)	
280 or 536	20	Buyer ID	
300 or 556	1	Invoice type	
301 or 557	1	Transaction type	
302 or 558	7	Invoice amount	
309 or 565	256	Digital signature of the above structure	
565 or 821	4	CRC	CRC is calculated from 0 to 565 or 821

			bytes.

Example 1 (without CRC):

Request: `88120400000000`

Response: *565 or 821 bytes* + `9000`

Example 2 (with CRC):

Request: `88120102000000`

Response: *569 or 825 bytes* + `9000`

Start Audit

Notifies the Secure element that the audit process has been initialized by E-SDC.

Secure element returns an encrypted message that shall be submitted to TaxCore as the content of the field `auditRequestPayload` of audit-proof request.

NOTE:
From the SE version 3.2.5, optionally, CRC can be calculated and used for data verification. If CRC is not used, the command is the same as in the previous applet version.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0 (no CRC)	`Case2Ext`	`0x88`	`0x21`	`0x040`	none	none	`0x000000`
>= 3.2.5 (with CRC)	`Case2Ext`	`0x88`	`0x21`	`0x010`	none	none	`0x000000`

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0 (no CRC)	*260 bytes data*	`0x9000`
>= 3.2.5 (with CRC)	*264 bytes data*	`0x9000`

Example 1 (without CRC):

Request: `88210400000000`

Response: *260 bytes data* + `9000`

Example 2 (with CRC):

Request: `88210102000000`

Response: *260 bytes data* + *4 bytes CRC data* + `9000`

End Audit

Notifies the Secure element that the audit process has been finalized by TaxCore. If APDU Command status is OK (0x90 0x00) consider the audit operation is completed.

NOTE:
From the SE version 3.2.5, optionally, CRC can be calculated and used for data verification. If CRC is not used, the command is the same as in the previous applet version.

APDU Request

SE Version	IsoCase	Class	Instruction	P1-P2	Command Length (Lc)	Command Data	Expected Length (Le)
>= 2.0.0 (no CRC)	`Case3Ext`	`0x88`	`0x20`	`0x040`	`0x000100`	*256 bytes received from TaxCore*	none
>= 3.2.5 (with CRC)	`Case3Ext`	`0x88`	`0x20`	`0x010`	`0x000104`	*256 bytes received from TaxCore + 4 bytes for CRC*	none

APDU Response

SE Version	Response Data	SW1SW2
>= 2.0.0 (no CRC)	none	`0x9000`

>= 3.2.5 (with CRC)	none	`0x9000`

Example 1 (without CRC):

Command Data:
`253AB91A21859A06813E8A880E10BA0C67A09DDBED0B7E001F638CA015D2E414744E0C5C2E0F5F827DFC`

Request:
`88200400000100253AB91A21859A06813E8A880E10BA0C67A09DDBED0B7E001F638CA015D2E414744E0C`
`9000`

Example 2 (with CRC):

Command Data:
`253AB91A21859A06813E8A880E10BA0C67A09DDBED0B7E001F638CA015D2E414744E0C5C2E0F5F827DFC`

Command Data CRC: `CEE700A0`

Request:
`88200102000104253AB91A21859A06813E8A880E10BA0C67A09DDBED0B7E001F638CA015D2E414744E0C`
`9000`

Secure Element Specific APDU Error Codes

This table contains the expected error codes and descriptions that a caller may encounter while working with the Secure Element Applet.

Error Code	APDU Command	Description	Error Code to POS
0x6301	Sign Invoice	PIN verification required before executing a command	1500
0x6302	Verify PIN	PIN verification failed – wrong PIN code	2100
0x6303	Verify PIN	Wrong PIN size	2100
0x6304	Sign Invoice	Maximum number of tax categories exceeded	SDC related
0x6305	Sign Invoice	Secure Element amount has reached the defined limit. The Secure Element is locked and no additional invoices can be signed before the audit is completed.	2210
0x6306	End Audit	End Audit is sent but there is no active Audit	SDC related

0x6307	Sign Invoice	Invoice fiscalization is disabled by system	2210
0x6310	Verify PIN	The number of allowed PIN entries exceeded	2110
0x63FF	Sign Invoice	A Secure Element counter has reached its limit. The Secure Element must be replaced.	SDC related
0x6700	End Audit	Data must be 256 bytes long	SDC related
0x6A80	End Audit	Proof of Audit command payload provided as APDU Command Data does not match the latest Start Audit one which Secure Element expects. Probably a new Start Audit was initiated after this one was ended.	SDC related
0x6A80	Sign Invoice	The tax category order id exceeds the maximum allowed for the Secure Element.	2310
0x6F00	End Audit	APDU Command Data cannot be recognized as a valid Proof of Audit	SDC related

File-Based Communication

Introduction

This section contains the description of the **File-based** communication with E-SDC.

File-based communication between TaxCore.API and E-SDC is foundation of Local Audit process.

General structure for storing files on removable memory units:

Removable memory root

- UID (folder)
    - audit packages (file)
    - UID.arp (file)
- UID.commands (file)
- UID.results (file)

NOTE:

The above structure displays files for all individual file transfers (copy audit files from E-SDC, upload commands to E-SDC, and transfer commands result from E-SDC). It is used here for explanatory purposes. Each individual file transfer includes only the files specific for that transfer.

Content

SD Cards or Flash Memory Drives Format

Each E-SDC shall work with the following file system formats of SD Cards and USB Flash drives:

- FAT
- FAT32
- NTFS

E-SDC is Configured Using Initialization Commands from an SD Card

Commands File

JSON file containing all pending commands must be stored in the root of the external disk volume and named **{UID}.commands** (e.g `D:\\YJ37C9Z9.commands`)

Command file may be generated and downloaded from either Taxpayer Admin Portal (by taxpayers) or the internal back office portal (by tax authority officers).

Commands File Format

Command file content must be formatted as valid JSON file.

```
[
    {
      "commandId": "GUID",
      "type": 0,
      "payload": "Command Specific Json as string",
      "uid": "string"
    }
]
```

End Of Line Characters

E-SDC must be able to process commands file using any standard EOL characters.

E-SDC Stores Audit Files on SD Card or USB Drive

An E-SDC shall perform an audit automatically once an SD Card or USB drive is inserted. If any commands are received on the same medium, they shall be executed **before** the proceeding with the Local audit.

NOTE:
Audit files must remain stored in E-SDC's local memory until it receives a Proof-of-Audit command from TaxCore.API

All files shall be stored in the folder(s) named after the UID(s) of the secure element(s) which created them.

Example: `G:\BJ3PN1S9\`, where G is the root of the SD card/USB drive and contains audit packages created by the secure element with the UID BJ3PN1S9.

If the folder(s) do not exist, an E-SDC shall create new one(s) - one for each UID.

All audit package files stored in E-SDC's local memory, no matter which secure element was used to create them shall be copied to the appropriate folder(s). Audit package files must be named using the following convention: `{UID}-{UID}-{Ordinal_Number}.json`.

Depending on whether the smart card secure element is connected to the E-SDC, the folder named after its UID may contain one `{UID}.arp` file. The content of the `{UID}.arp` file is described in Submit Audit Request Payload - ARP.

File structure for this transfer should look like this:

Removable memory root

- UID folder
    - audit package files
    - UID.arp

NOTE:
Dumped audit package files must be created by secure element(s) from the appropriate target environment, e.g. if the Local Audit is performed for the production environment, only audit package files created by the production environment secure elements can be dumped on an SD Card or USB Drive. This is done in order to avoid uploading audit packages to environments which do not recognize the UIDs created for different environments.

E-SDC Executes Commands Received via SD Card or USB Drive

An E-SDC shall process commands automatically upon insertion of SD Card or USB Flash drive. Command execution takes precedence over a Local audit.

JSON file containing all pending commands must be stored in the root of the external disk volume and named **{UID}.commands** (e.g `D:\\YJ37C9Z9.commands`).

File structure for this transfer should look like this:

Removable memory root

- UID.commands

E-SDC shall execute only those commands with the same UID as UID assigned to the digital certificate of the Secure Element (stored in the *SerialNumber* field of the certificate subject).

Command types and the structure are explained in section Commands.

E-SDC Stores a Command Execution Result to the SD Card or USB Drive

After commands have been executed, E-SDC must generate a JSON file named **{UID}.results**. The result must be in format described in paragraph *Commands Execution Results* in Commands. It is stored in the root folder on the SD Card/USB drive.

Example: **G:\BJ3PN1S9.results**

where **G is the root of the SD card/USB drive** and **BJ3PN1S9 is an example UID** of the smart card in use

If file with the same name exists on the SD Card/USB drive, it must be overwritten.

File structure for this transfer should look like this:

Removable memory root

- UID.results

Manuals

E-SDC must have a user manual that explains the following topics in detail:

1. System requirements
2. Installation instructions for the technicians performing the installation and integration of an E-SDC device or software at a sales point
3. Properly connecting the E-SDC to a POS
4.

User instructions for the operator (cashier or shopkeeper) explaining normal operations in detail

- Information about the supported card readers or integrated card reader
- How commands are received and executed via USB drive or SD card
- How to obtain help/support regarding the product
- Steps for troubleshooting
- How the product license is activated
- Local and/or remote audit instructions
- How to properly shut down or uninstall the product

5. A detailed explanation (with visuals) for each product notification
6. Explanation of each custom error (manufacturer-specific)