

# ZENTRALER KREDITAUSSCHUSS

MEMBERS: BUNDESVERBAND DER DEUTSCHEN VOLKSBANKEN UND RAIFFEISENBANKEN E.V. BERLIN • BUNDESVERBAND DEUTSCHER BANKEN E. V. BERLIN • BUNDESVERBAND ÖFFENTLICHER BANKEN DEUTSCHLANDS E. V. BERLIN • DEUTSCHER SPARKASSEN- UND GIROVERBANDE. V. BERLIN-BONN • VERBAND DEUTSCHER HYPOTHEKENBANKENE. V. BERLIN

## Implementation Guide

### EBICS

### *(Electronic Banking Internet Communication Standard)*

(Supplement to current  
“DFÜ-Abkommen”)

**Version 1.7 of September 1st, 2008**

**Final Version**

---

## Amendment history

Version	Author	Date	Comments
1.0	Michael Krüger, Dr Ralph Peters, Karin Rosenauer	28.02.2005	Creation of the document
1.1	Michael Krüger, Dr Ralph Peters, Karin Rosenauer	28.07.2005	Incorporation of comments on Version 1.0, introduction of order type "HSA"
1.2	Michael Krüger, Dr Ralph Peters, Karin Rosenauer	19.08.2005	Incorporation of comments on Version 1.1
1.3	Michael Krüger, Dr Ralph Peters, Karin Rosenauer	26.08.2005	Incorporation of comments on Version 1.2
1.4	Sabine Wenzel	11.08.2006	Additional chapter 5.4.3, incorporation of comment on the number of parallel customer sessions to be restricted. In respect of content, the version 1.4 of the Implementation Guide refers to the EBICS detailed concept version 2.1.
1.5	Sabine Wenzel	22.03.2007	Note on the suspension request in chapter 5.3 Note on the provision of end-of-record characters on part of the bank and on the order of processes in chapter 5.4.3 Note on the composition of hash values of the financial institution's public keys X001 and E001 in chapter 9.5 as well as incorporation of an appendix containing an example on the composition of a hash value (additional chapter 13 "Appendices")  In respect of content, the version 1.5 of the Implementation Guide refers to the EBICS detailed concept version 2.2.
1.6	Sabine Wenzel	04.10.2007	In chapter 5.4.3 the following details concerning implementations on the bank's end were supplemented: <ul style="list-style-type: none"><li>• Length of lines in the customer protocol</li><li>• Handling of orders not completely authorized</li><li>• Line breaks</li></ul>
1.7	Sabine Wenzel	15.05.2008	Additional appendix 14.2 with examples to clarify the term "technical subscriber". Shift of existing informations from chapter 5.3.3 and 5.4.3 to the new chapter 12 due to the fact that they were misarranged as to content.  As regards content, the version 1.7 of the Implementation Guide refers to the EBICS Detailed Concept version 2.4.

## Contents

<b>1</b>	<b>Introduction .....</b>	<b>6</b>
<b>2</b>	<b>Symbology and data types .....</b>	<b>8</b>
2.1	Symbology .....	8
2.2	Data types .....	9
<b>3</b>	<b>Architecture &amp; components of the EBICS applications.....</b>	<b>11</b>
3.1	Requirements of the EBICS applications .....	11
3.1.1	Requirements of EBICS clients (customer systems).....	11
3.1.2	Requirements of EBICS servers.....	11
3.2	Example architecture.....	12
3.3	EBICS clients.....	14
3.4	Basic components and services .....	15
3.5	Persistent data.....	16
<b>4</b>	<b>Implementation instructions for the identification and authentication signature.....</b>	<b>18</b>
4.1	XML signature in the EBICS context .....	18
4.2	Composition of the XML signature structure .....	21
4.3	Filling out the XML signature structure .....	24
4.4	Meaning of the XPointer expression in <code>ds:Reference@URI</code> .....	26
4.5	Verification of the identification and authentication signature.....	26
<b>5</b>	<b>Replay avoidance using Nonce and Timestamp .....</b>	<b>29</b>
5.1	Process description .....	29
5.2	Formats of “Nonce” and “Timestamp” .....	30
5.2.1	Format of “Nonce” .....	30
5.2.2	Format of “Timestamp” .....	30
5.3	Actions of the customer system.....	31
5.3.1	Filling out the fields “Nonce” and “Timestamp” .....	31
5.3.2	Behaviour in the event of error message „EBICS_TX_MESSAGE_REPLAY“ .....	31
5.4	Actions of the bank system.....	32
5.4.1	Checking “Nonce” and “Timestamp” .....	32
5.4.2	Storage of “Nonce”/“Timestamp” pairs .....	33
5.4.3	Further Recommendations .....	34
<b>6</b>	<b>XML parsers and generators .....</b>	<b>35</b>
6.1	Tasks .....	35

# EBICS specification

EBICS – Implementation Guide Version 1.7

---

6.2	Models .....	35
6.3	Selection of a suitable model.....	37
<b>7</b>	<b>Random numbers .....</b>	<b>38</b>
7.1	Cryptographically-strong pseudo-random numbers .....	38
7.2	Java class SecureRandom.....	39
<b>8</b>	<b>Transaction management.....</b>	<b>40</b>
8.1	Transaction management at the customer's/subscriber's end .....	40
8.1.1	Replay avoidance .....	40
8.2	Transaction management at the bank's end .....	40
8.2.1	Lifespan of an EBICS transaction.....	40
8.2.2	Transaction states .....	41
8.2.2.1	State diagram of an upload transaction .....	41
8.2.2.2	State diagram of a download transaction .....	45
<b>9</b>	<b>Key management.....</b>	<b>50</b>
9.1	Generation of the RSA keys .....	50
9.2	Generation of the symmetrical 3DES key.....	50
9.3	Key storage .....	50
9.3.1	Standard formats .....	51
9.3.1.1	PKCS#12 .....	51
9.3.1.2	XML .....	51
9.3.2	Standard interfaces for accessing keystores.....	52
9.3.2.1	PKCS#11 (Cryptographic Token Standard Interface).....	52
9.3.2.2	Microsoft CryptoAPI.....	52
9.3.2.3	Java APIs.....	52
9.3.3	Alternatives for key storage .....	53
9.3.3.1	Private keys .....	53
9.3.3.2	Public keys.....	54
9.4	Subscriber initialisation.....	55
9.5	Verification of the bank keys.....	55
9.6	Amendment of the subscriber keys .....	57
<b>10</b>	<b>Information on segmentation .....</b>	<b>58</b>
<b>11</b>	<b>Recommendations for filling out fields in the EBICS protocol.....</b>	<b>59</b>
11.1	Manufacturer specifications for the customer product.....	59

<b>12</b>	<b>Further recommendations</b> .....	<b>60</b>
12.1	Actions of the customer system.....	60
12.2	Actions of the bank system.....	60
<b>13</b>	<b>Utilised standards and references</b> .....	<b>61</b>
<b>14</b>	<b>Appendices</b> .....	<b>62</b>
14.1	Appendix 1 – Example of the computation of a hash value.....	62
14.2	Appendix 2 – Further Examples .....	67
14.2.1	Clarification of the Term “Technical Subscriber“.....	67
14.2.2	Example for the Interpretation of Field AccountInfo@ID in the Order Types HKD and HTD.....	69

## 1 Introduction

The Electronic Banking Internet Communication Standard (EBICS) expands the existing „DFÜ-Abkommen“<sup>1</sup> by the functionality of multi-bank capable, secure communication via the Internet.

In doing this, the application-orientated elements of the „DFÜ-Abkommen“, i.e. the multi-bank capability, the transmission of files in bank-technical formats using order types and the defined security processes for electronic signatures (ES), are retained in their entirety. The order types and the electronic signature that are defined in the „DFÜ-Abkommen“ are supported in Version A004 and above.

At the application level, the process “Remote data transmission with customer” is augmented by the concept of Distributed Electronic Signature (VEU), which allows chronologically and spatially-independent authorisation of orders via the financial institution’s server.

This document, the “EBICS Implementation Guide”, is based on the “EBICS detailed concept”. In addition to the stipulations and guidelines specified in the detailed concept, the “EBICS implementation guide” will provide information on selected aspects of the implementation process and will point out implementation alternatives.

Information on remote data transmission order types can be found in the „DFÜ-Abkommen“. Basically, all guidelines and definitions of the „DFÜ-Abkommen“ will apply insofar as the converse is not specified in the EBICS documentation (detailed concept, implementation guide).

The EBICS implementation guide has the following structure:

- Chapter 2 provides an introduction to the symbology and the data types for the XML schema.
- Chapter 3 contains an example architecture and lists software base components and services.
- Chapter 4 shows implementation details for the identification and authentication signature. The structure of “XML signature” is handled in the same way as the filling out of the individual elements of this structure as expected in the EBICS context.
- Chapter 5 is dedicated to the avoidance of replay attacks. The “Nonce”/“Timestamp” schema that was developed for this purpose is explained, and information is given on the correct handling of these parameters (from the viewpoint of both the customer system & the bank system).
- Chapter 6 lists different models of XML parsers and generators, together with their respective advantages and disadvantages.

---

<sup>1</sup> See EBICS detailed concept for dependencies on the Remote Data Transmission Agreement.

## **EBICS specification**

EBICS – Implementation Guide Version 1.7

---

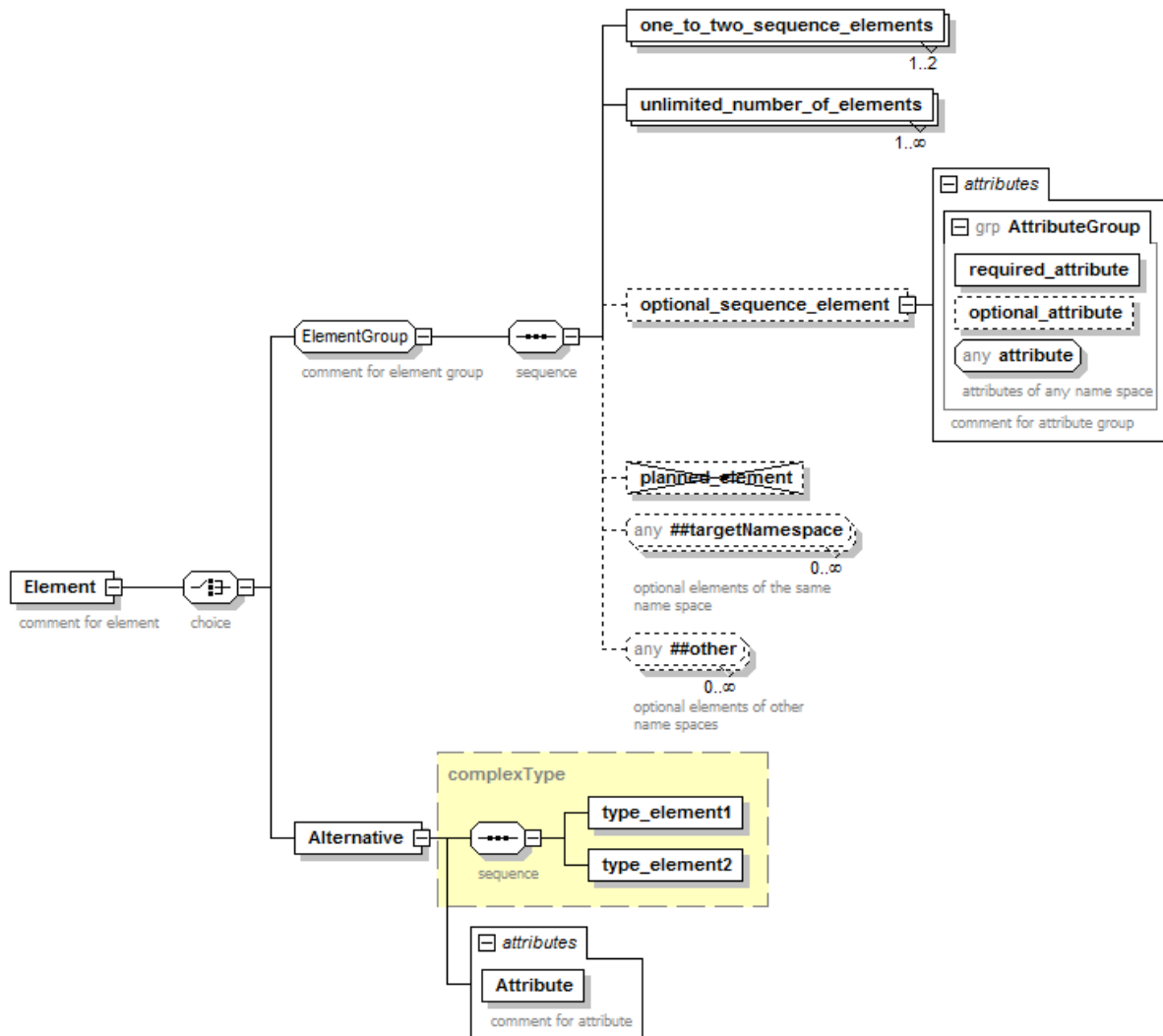
- Chapter 7 is concerned with the generation of cryptographically-strong pseudo-random numbers.
- Chapter 8 describes aspects of the implementation and transaction management at the customer's and financial institution's end.
- Chapter 9 looks at the theme of key management, including the aspects of key generation and storage.
- Chapter 10 elucidates the procedures for the segmentation of order data.

## 2 Symbology and data types

### 2.1 Symbology

The following symbology is used for graphical representation of XML schemas:

- Elements are placed in rectangles
- Attributes are also placed in rectangles and are surrounded by an “attributes” box
- Elements, attributes and other declarations that belong to a complex type are surrounded by a dashed box that is highlighted in yellow
- A “branch” (corresponds to `choice` in XML schema) is shown as an octagon containing a switch symbol for three possible switch positions. The connecting lines to the possible alternatives branch out on the right of the symbol.
- A “sequence” (corresponds to `sequence` in XML schema) is shown as an octagon containing a line symbol with three points on it. The connecting lines to the individual sequence elements branch out on the right of the symbol.
- Symbols with solid edges denote mandatory use, and in the XML schema correspond to the attribute `minOccurs="1"` for elements or `use="required"` for attributes.
- Crossed-out symbols denote optional usage, and in the XML schema correspond to the attribute `minOccurs="0"` for elements or `use="optional"` for attributes
- Crossed-out symbols denote planned usage, and in the XML schema correspond to the attribute combination `minOccurs="0" maxOccurs="0"` for elements or `use="prohibited"` for attributes
- “m..n” in the right lower corner of an element symbol restrict the use of the element to m- to n-times occurrence, and in the XML schema correspond to `minOccurs="m" maxOccurs="n"`; correspondingly, where “m..∞” `minOccurs="m" maxOccurs="unbounded"`
- Element groups are represented by octagons, and correspond to the `group` declaration in the XML schema
- Attribute groups are surrounded by boxes with the respective group names and correspond to the `attributeGroup` declaration in the XML schema.



**Diagram 1: XML schema symbols**

## 2.2 Data types

The XML schema defines a set of primitive and derived data types that can be used to form your own data types.

The following primitive data types are primarily used in conjunction with EBICS:

- **string**: String of characters with unrestricted length and any structure
- **Boolean**: boolean truth value with the characteristics “true” (=1) or “false” (=0)
- **decimal**: decimal numbers to any degree of accuracy
- **dateTime**: time stamp with date and time in accordance with ISO 8601
- **date**: date in accordance with ISO 6801
- **hexBinary**: hexadecimal value with unrestricted length
- **base64Binary**: data type to record base64-coded binary data

## EBICS specification

- **anyURI**: uniform resource locator (e.g. URL, IP address).

The following pre-defined data types are derived from primitive data types and are used in the EBICS standard:

- **normalizedString**: string of characters that has spaces (blanks) removed at the start and end
- **token**: a normalizedString that contains no line feeds and no multiple spaces in succession
- **language**: language tags in accordance with RFC 1766
- **nonNegativeInteger**: non-negative integer values
- **positiveInteger**: positive integer values.

With the help of the aforementioned data types, new data types are defined in the EBICS schema:

- **simple data types** merely define restrictive or expanding characteristics with regard to the value range of an existing primitive or derived data type, i.e. they derive from an existing data type
- **complex data types** define new structures composed of fields and attributes of different (simple or complex) data types.

### **3 Architecture & components of the EBICS applications**

#### **3.1 Requirements of the EBICS applications**

In this chapter, requirements will be listed that must be fulfilled before implementation of an EBICS application. A differentiation is drawn between general requirements that relate to the infrastructure at the client's or server's end, and requirements that are specific to EBICS. These requirements are to be observed when the financial institutions implement their own individual security concepts/considerations.

##### **3.1.1 Requirements of EBICS clients (customer systems)**

- Key management requirement (see also Chapter 9.3):
  - Private subscriber keys are protected against unauthorised readout and modification
  - The bank's public keys are protected against unauthorised modification
  - Secret symmetrical keys are protected against unauthorised readout and modification
  - Guidelines for the secure storage of private/public keys are a part of the financial institutions' customer conditions.
- TLS certificate management requirement:
  - The certificate that is used as a trust anchor when examining the bank's TLS certificate is protected against unauthorised modification/replacement.
- General requirements on the infrastructure:
  - The customer software is secured against any manipulation that could mislead the subscriber as to the progress of EBICS transactions.
  - The internal communication channels for unencrypted bank-technical data are protected against interception and manipulation.
  - The internal communication channels for EBICS messages are protected against interception and manipulation.
  - Each customer is generally responsible for providing an individual solution for the protection of their customer software and internal communication channels.

##### **3.1.2 Requirements of EBICS servers**

- Key management requirements (see Chapter 9.3):
  - The subscriber's public keys are protected against unauthorised modification
  - The bank's private keys are protected against unauthorised readout and modification
  - If the symmetrical keys (encrypted or in clear text) are stored, they must be protected from unauthorised readout and modification.

## EBICS specification

- In the event of unencrypted transmission of bank-technical order data and bank-technical signatures on the bank system's internal communication channels, the confidentiality of this data must be sufficiently ensured.
- Important order details such as e.g. order type, order number or order parameters are also to be protected against manipulation after their extraction from EBICS messages.
- Processes for authorisation verification, bank-technical preliminary checking and the execution of business transactions are to be sufficiently protected against manipulation.
- Secure storage of subscriber data in the subscriber administration system.
- Secure storage of transaction data in the EBICS transaction data administration system.

### 3.2 Example architecture

Diagram 2 contains an example architecture of a multi-layer distributed application for realisation of the EBICS protocol.

Alternatives for realisation of the client layer are covered in Chapter 3.3. The focus of the example architecture is realisation of the EBICS server end. The structure of the business or EIS (Enterprise Information System) layer from modules only represents the substantial functional building blocks of an EBICS server.

On the one hand, these modules provide functionalities that are not required in the FTAM process, and for this reason must be newly-implemented for EBICS. Examples of such modules are: EBICS processor, message authentication, EBICS transaction management, VEU management and those parts of the module "verification and implementation of orders" that relate to new order types.

On the other hand, modules are realised by delegation to existing applications/modules from the FTAM process that are placed in the EIS layer in the diagram. Examples of such modules that are to be incorporated in the EBICS application are "order authorisation" or "verification and implementation of bank-technical orders".

The responsibilities of the business layer modules are described briefly in the following text:

- **EBICS processor**  
The EBICS processor is responsible for the processing of incoming EBICS requests and generation of the associated EBICS responses in collaboration with the components "message authentication", "subscriber authorisation" and "EBICS transaction management".
- **Message authentication**  
The module "message authentication" is responsible for the generation/verification of the identification and authentication signatures of the EBICS messages.
- **Subscriber authorisation**  
Subscriber authorisation is responsible for the order type authorisation of the initiating party via EBICS.

- **EBICS transaction management**

This module is responsible for the management of the EBICS transactions in the bank system. It is responsible for administration of the state of open EBICS transactions and the intermediate storage of the transmitted user data segments and ES's.

System-technical orders, download orders and VEU orders are carried out synchronously and for this reason are also forwarded directly to the "verification and processing of orders" module.

On the other hand, bank-technical upload orders are carried out asynchronously and are initially forwarded to the "management of open transactions" module.

- **Management of pending orders**

The module "management of pending orders" is responsible for the asynchronous processing of orders. Sufficiently-authorized orders are forwarded to the "**management of pending authorised orders**", verification of insufficiently-authorized orders is initially passed on to the component "**VEU management**".

- **"VEU management"**

This module manages insufficiently-authorized orders. In doing this, "cross-customer authorisation" should also be noted: Example: A subscriber of customer A is authorised to provide an ES for payments of customer A, a subscriber of customer B is also authorised to provide an ES for payments of customer A. As soon as an order within the VEU management is sufficiently authorised, it is assigned to the "management of pending authorised orders".

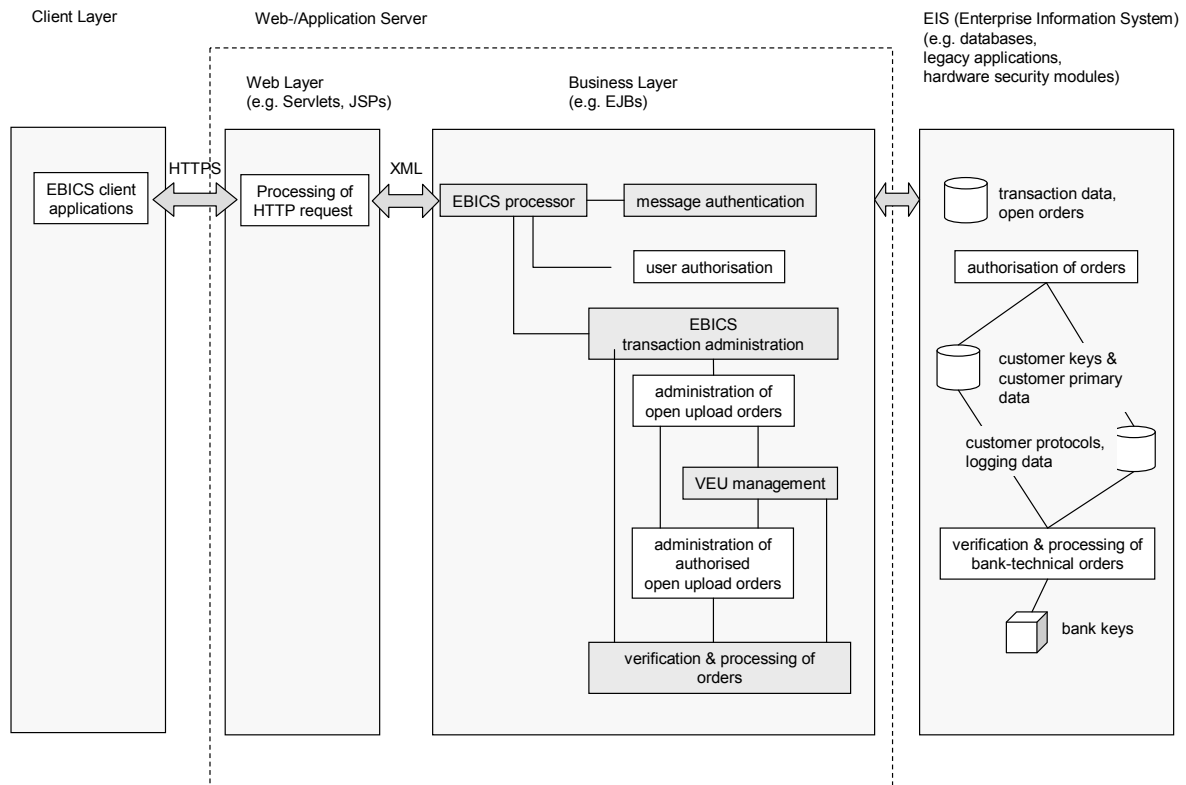
- **Verification and processing of orders**

This module is responsible for the verification and processing of orders. The verification and processing of bank-technical orders takes place via access to information systems in which e.g. subscriber data and keys are stored, and by delegation to existing applications/modules.

The processing of VEU orders (e.g. submission of a bank-technical ES, order cancellation) takes place in co-operation with the component "VEU management".

# EBICS specification

## EBICS – Implementation Guide Version 1.7



**Diagram 2: Example architecture of an EBICS application**

### 3.3 EBICS clients

The main responsibilities of EBICS clients are:

- Execution of EBICS transactions
  - Generation of A004 signatures
  - Generation of EBICS requests
  - Processing of EBICS responses
- Support of VEU via provision of workflows from HVU, HVD, HVT
- Key & subscriber management at the customer's end

EBICS clients can be categorised as follows:

- **Java applet solutions**

The implementation environment of Java applet solutions is a web browser and a Java VM. At run time, application-specific code is sent to the client and implemented locally. In comparison with HTML clients, applet solutions support better GUI's and improved user interaction with the help of Java AWT/Swing.

In EBICS applications, applets can take on the role of the EBICS client. The size of the applet should be taken into consideration when deciding upon the use of an applet solution.

- **Desktop applications (Rich/Fat Clients)**, that must be installed by the individual subscribers.

In the case of Java applications, the use of Java WebStart for distribution of the application can minimise the amount of administration required.

- **Client/server application as a customer system**

Note: In the following description, “client” denotes the client part of the customer system and “server” denotes the server part of the customer system. Together, client and server comprise the EBICS client.

Beyond the representation of the GUI and user interaction, the minimum requirement for the client comprises the generation of A004 signatures using the secret bank-technical subscriber key (stored at the subscriber’s end).

For this reason, HTML clients (thin clients that, along with a web browser as an implementation environment, do not require application-specific code) are unsuitable for the task. On the other hand, applet solutions where applets are responsible for the generation of the A004 signatures are suitable as clients.

If it is necessary to transmit order data from the client to the server, this should take place via rapid, reliable communication channels (e.g. Intranet) on which the restrictions of EBICS regarding segment size do not apply.

The remainder of the responsibilities of the customer system can be transferred to the server of the customer system. Central services such as e.g. management of the public bank keys, management of EBICS subscriber data, EBICS customer system transaction management, etc. can be outsourced onto the server.

### 3.4 Basic components and services

The following basic components are required for implementation of the EBICS standard:

- XML generator, XML parser, XML schema validator for generation/validation of EBICS messages
- XML signature tool for generation/validation of the identification and authentication signatures
- Communication software for HTTPS
- Data compression software: ZIP (in accordance with RFCs 1950 & 1951)
- Data coding software: base64 encoder/decoder (in accordance with RFCs 1421 & 2045)
- Crypto-components (optionally also implemented as hardware) for execution of the following tasks:
  - Generation of asymmetrical RSA keys
  - Generation of symmetrical 3DES keys
  - Encryption/decryption in accordance with the X001 process (based on 2-key triple DES)
  - Generation/verification of A004 signatures (based on RSA signatures, RIPEMD-160)
  - Hash functions: SHA-1

- Pseudo-random number generators for Nonces and EBICS transaction IDs.

Java solutions are available for components that implement internationally-recognised standards.

Provision of the following services is required for transparency of processes on the part of the EBICS server:

- Logging for tracing EBICS messages  
Realisation of the logging service, especially the format of the logging data, is not a part of the EBICS standard. This function is to be implemented individually for each financial institution.
- Protocol service for recording customer actions in accordance with the EBICS detailed concept.

### 3.5 Persistent data

The following data must be persistently provided by the customer and bank systems for communication via EBICS:

- **Persistent data in the customer system:**
  - Subscriber data (subscriber ID, customer ID) for identification to the financial institutions  
The persistent storage of subscriber data is primarily expedient if the administration of subscriber data takes place centrally in the customer system.
  - Public and private subscriber keys (encryption, identification and authentication, signature)
  - Public keys (encryption, identification and authentication) of the financial institutions with which communication is to take place via EBICS
  - Bank parameters of the financial institutions (see order type “HPD”) including the connection data (URL) for the individual financial institutions.
- **Persistent data in the bank system:**
  - Master data of the registered subscribers (IDs, addresses, authorisations; see order type “HTD” in the EBICS detailed concept)
  - Master data of the registered customers (IDs, addresses, authorisations; see order type “HKD” in the EBICS detailed concept)
  - Public and private keys (encryption, identification and authentication) of the financial institution
  - Public subscriber keys (encryption, identification and authentication, signature)
  - Bank parameters, insofar as these support retrieval from the bank system (see order type “HPD” in the EBICS detailed concept), host ID
  - List of the “Nonce”/“Timestamp” pairs (see Chapter 5.4.2)
  - Data of the transaction management, VEU processing (see EBICS detailed concept, Chapter 8), customer protocols, logging data.

**EBICS specification**

EBICS – Implementation Guide Version 1.7

---

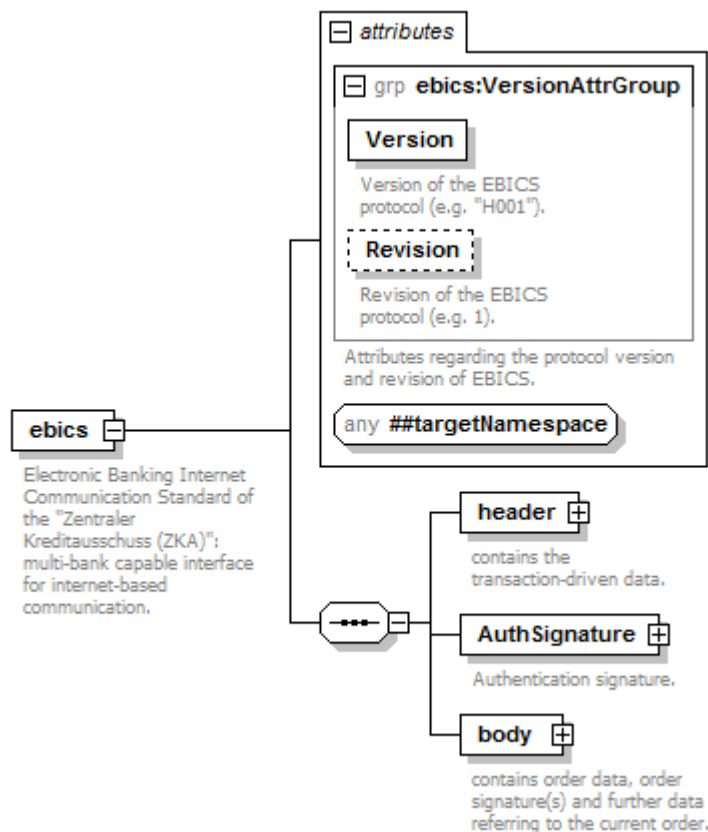
## 4 Implementation instructions for the identification and authentication signature

The format “XML signature” in accordance with RFC 3275 is used for the EBICS identification and authentication signature in Version “X001”. The current version of XML signature is documented on the W3C website <http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>. The schema for XML signature itself can be found at <http://www.w3.org/2000/09/xmlsig#> (“xmlsig\_core\_schema.xsd”).

An example is given in Appendix 1 (see also chapter 13).

### 4.1 XML signature in the EBICS context

The EBICS structure element for the identification and authentication signature is called `ebics/AuthSignature` and can be found in the EBICS schemas “`ebics_request.xsd`” und “`ebics_response.xsd`” in an enumeration sequence between the EBICS header and body.



**Diagram 3: Identification and authentication signature in the EBICS context**

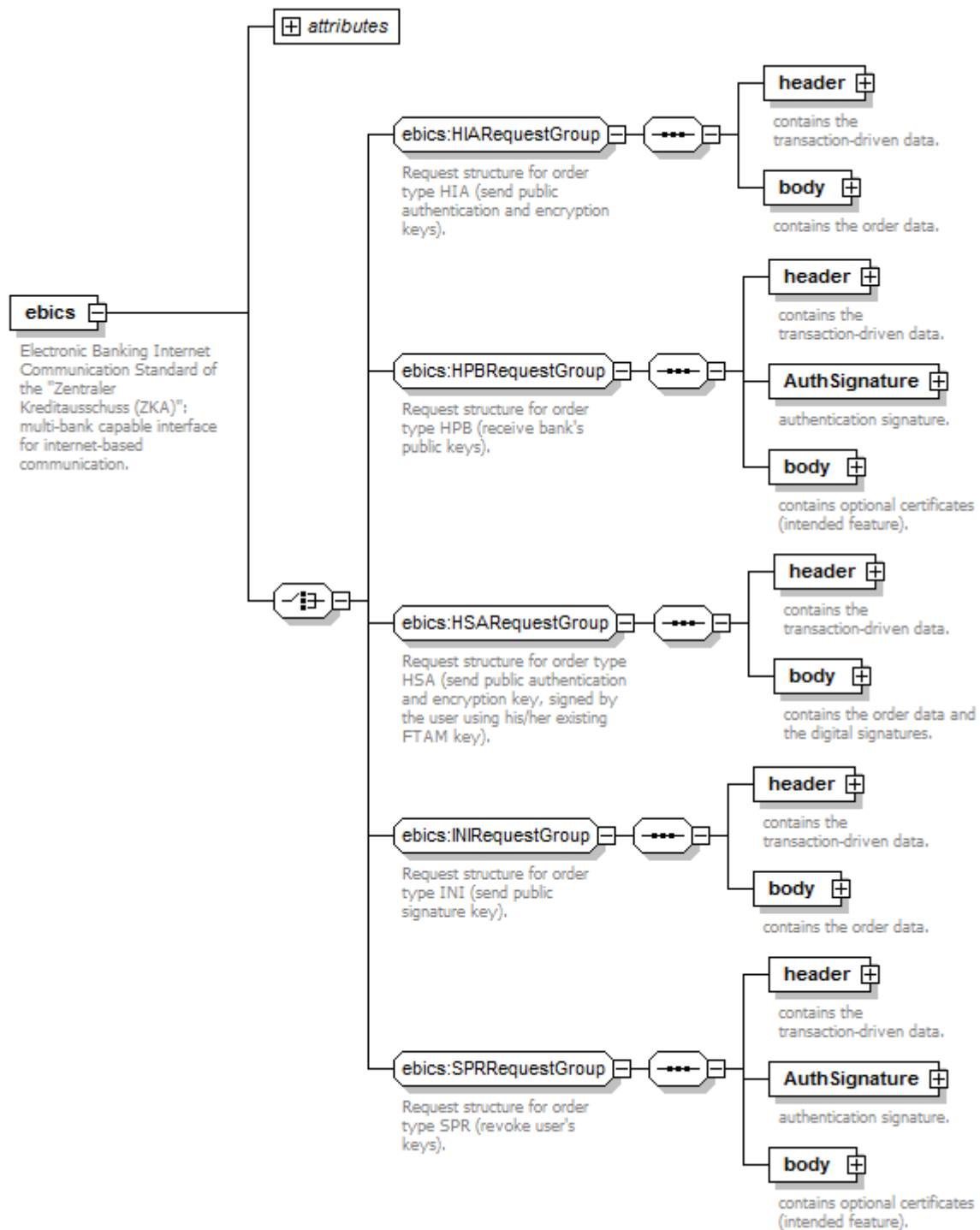
The EBICS schemas “`ebics_keymgmt_request.xsd`” and “`ebics_keymgmt_response.xsd`” are borrowed from the aforementioned standard EBICS schemas. They are used in the following key management order types:

## EBICS specification

- “INI” (subscriber initialisation: sending the public bank-technical key)
- “HIA” (subscriber initialisation: sending the public identification and authentication keys and the encryption keys)
- “HSA” (subscriber initialisation: sending the public identification and authentication keys and the encryption keys, bank-technically signed with the existing FTAM signature key)
- “HPB” (download the financial institution’s public key)
- “SPR” (suspend the subscriber’s keys).

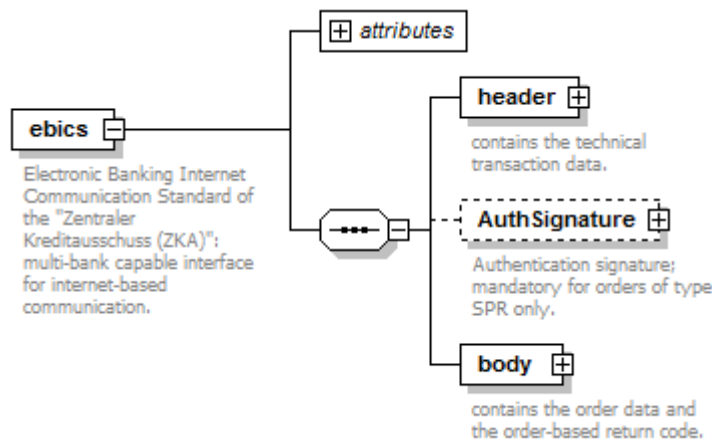
For the EBICS key management schema “ebics\_keymgmt\_request.xsd”, only order types “HPB” and “SPR” require an identification and authentication signature (“INI”, “HIA” and “HSA” do not), in the case of the EBICS key management schema “ebics\_keymgmt\_response.xsd” the identification and authentication signature is only mandatory for the order type “SPR” (not for “INI”, “HIA”, “HSA” and “HPB”).

For this reason, in the case of “ebics\_keymgmt\_request.xsd” a separate element group is responsible for each order type that firmly contains the selected order type and, depending on requirements, lists the identification and authentication signature either mandatorily or not at all (see Diagram 4).



**Diagram 4: Identification and authentication signature in the EBICS key management context (request)**

In the case of the EBICS schema "ebics\_keymgmt\_response.xsd", such grouping in accordance with order types is not carried out. Instead, with this schema the element ebics/AuthSignature is generally not constructed in a binding manner (see Diagram 5).



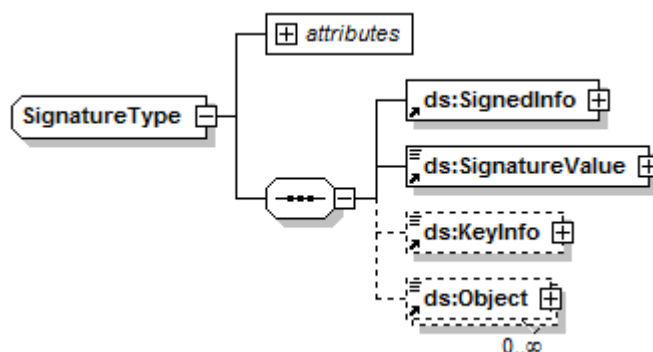
**Diagram 5: Identification and authentication signature in the EBICS key management context (response)**

In the case of “INI”, “HIA” and (for the direction of responses) “HPB”, the necessary public keys of the customer and the financial institution for verifying an identification and authentication signature are not yet known or are not yet verified. Generation and verification of the identification and authentication signature of these order types is therefore pointless or even impossible.

On the other hand, in the case of order type “SPR”, which also imperatively requires an identification and authentication signature in the response, an existence check on the identification and authentication signature is also necessary over and above schema validation of the received message. The schema validation alone is not sufficient due to the unbinding layout of the identification and authentication signature.

## 4.2 Composition of the XML signature structure

The element `AuthSignature` is of type `ds:SignatureType` (with `xmlns:ds="http://www.w3.org/2000/09/xmldsig#"` ), which has the following substructure:

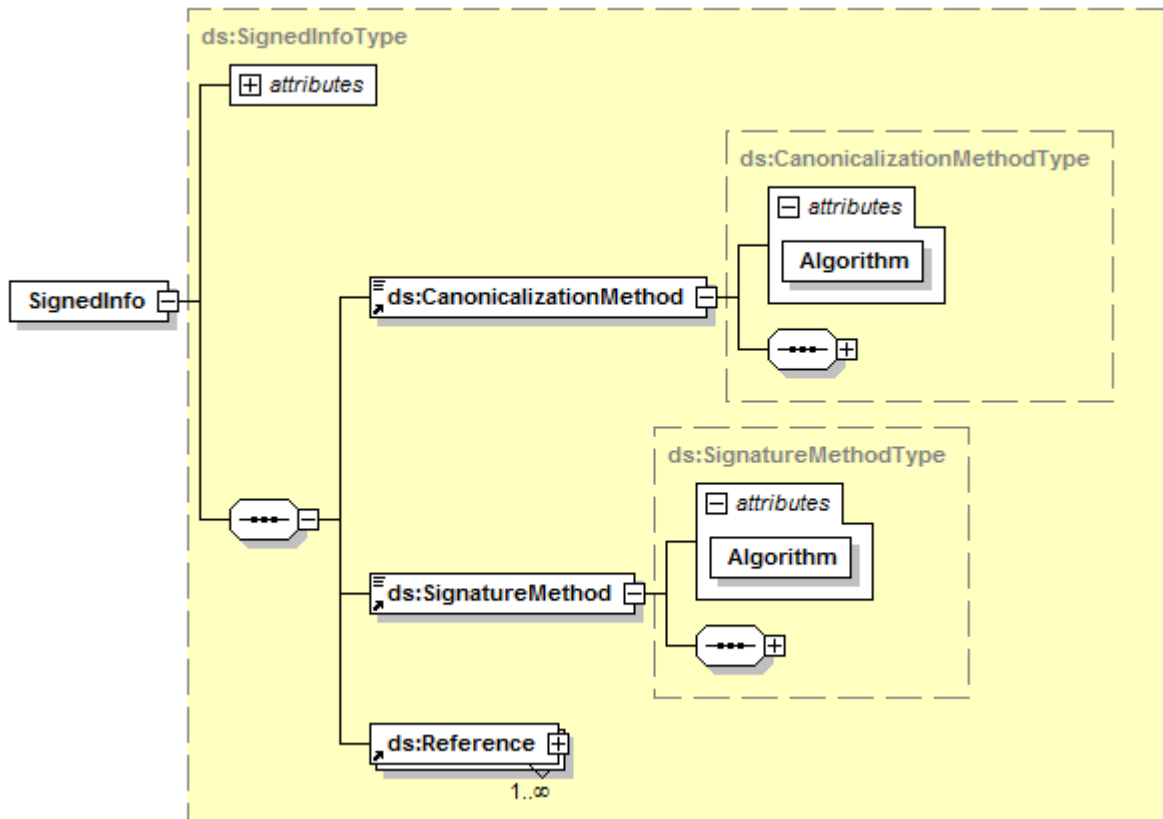


**Diagram 6: Type "SignatureType" of the XML signature**

## EBICS specification

EBICS – Implementation Guide Version 1.7

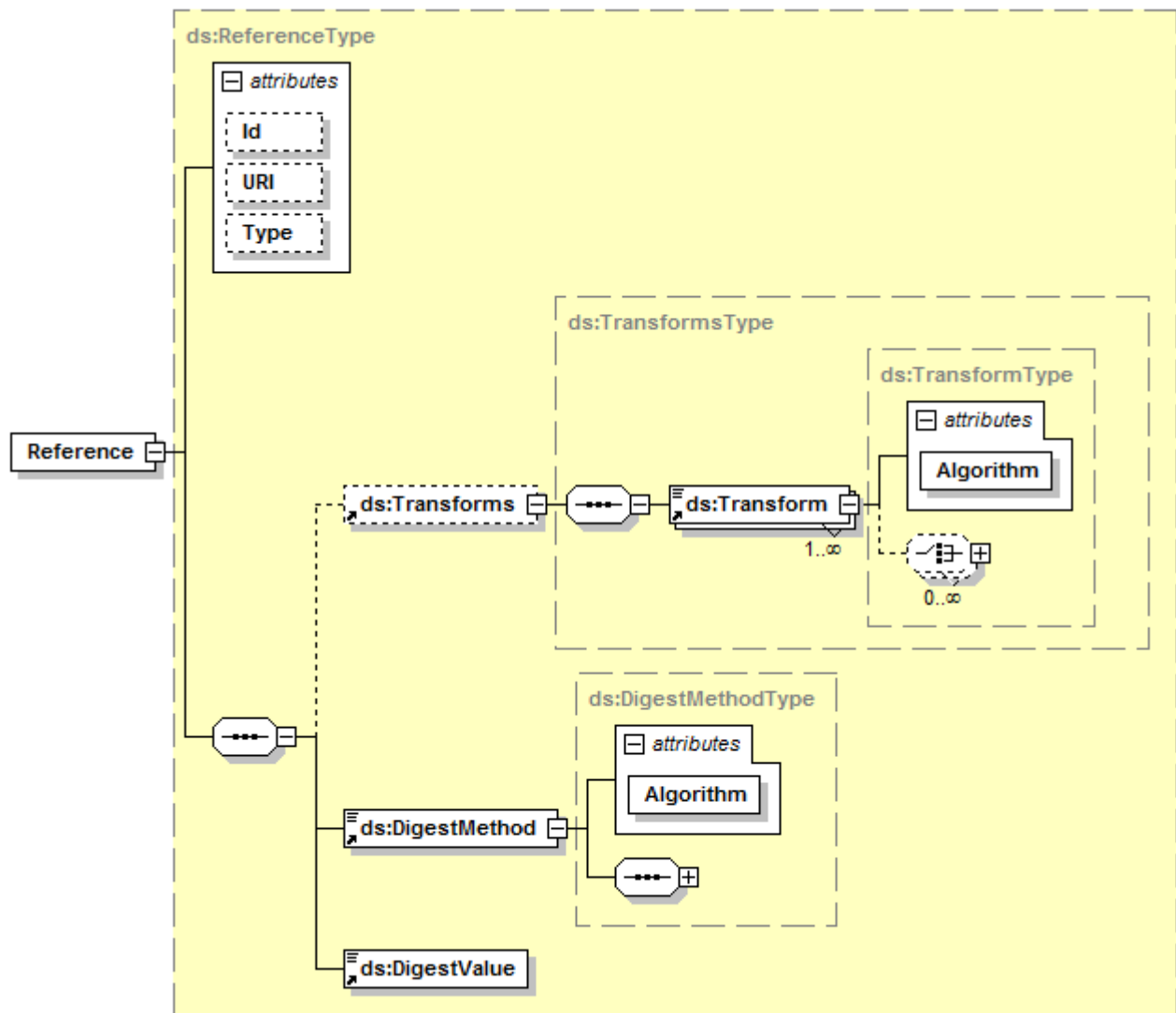
The element `ebics/AuthSignature/ds:SignedInfo` contains parameters for the XML signature, but not the signature itself. This is stored in the element `ebics/AuthSignature/ds:SignatureValue`.



**Diagram 7: Element “SignedInfo” of the XML signature (from “AuthSignature” of type “SignatureType”)**

The following parameters of `ds:SignatureType` are required in connection with EBICS:

- Canonisation algorithm (`ds:CanonicalizationMethod`): Here, the algorithm is specified in the attribute `@Algorithm`, in the form of a URI, that is to canonise the `ds:SignedInfo` structure itself, i.e. convert it into a standardised form, before it is used for signature configuration
- Signature algorithm (`ds:SignatureMethod`): Using the attribute `@Algorithm`, at this point the algorithm is specified in the form of a URI to configure and verify the signature.



**Diagram 8: Element “Reference” of the XML signature (from “SignedInfo”)**

- Reference of the data that is to be signed (`ds:Reference`): Again, this element contains a substructure with the help of which a description can be given for the data that is to be signed and its hash value:
  - `@URI` references the data structure that is to be signed.
  - In the attribute `@Algorithm` of the multiple sub-element `ds:Transform`, `ds:Transforms` specifies, in the form of a URI, the algorithms with the help of which the referenced data are to be transformed before calculation of the hash value.
  - In the attribute `@Algorithm`, `ds:DigestMethod` names, in the form of a URI, the algorithm for hash value configuration via the transformed data.
  - `ds:DigestValue` finally contains the hash value for referenced, transformed data in base64 coding.

### 4.3 Filling out the XML signature structure

The aforementioned XML fields for configuration of the identification and authentication signature in Version “X001” are to be filled out in accordance with the following specifications:

- `ds:CanonicalizationMethod@Algorithm`: The algorithm “Canonical XML” (“<http://www.w3.org/TR/2001/REC-xml-c14n-20010315>”) is to be used here.
- `ds:SignatureMethod@Algorithm`: RSA with SHA-1 (“<http://www.w3.org/2000/09/xmldsig#rsa-sha1>”) serves as an algorithm pair for configuration of the identification and authentication signature.
- `ds:Reference@URI`: The reference, in the form of a URI, must cover all elements & their sub-structures that have occupied the attribute `@authenticate` with the value `true`. `@URI="#xpointer(//*[ @authenticate='true'])"`.
- `ds:Reference/ds:Transforms/ds:Transform@Algorithm`: “Canonical XML” is again to be used as a transformation algorithm for the referenced XML data. Other transformations are not envisaged.
- `ds:Reference/ds:DigestMethod@Algorithm`: SHA-1 (“<http://www.w3.org/2000/09/xmldsig#sha1>”) serves as the hash algorithm for the transformed XML data.
- `ds:Reference/ds:DigestValue`: The result of the following operations is to be entered in this field:
  - Transformation of the referenced XML data in accordance with “Canonical XML”
  - Hash value configuration of the transformed XML data via SHA-1
  - Encoding of the hash value with base64.

In Version “X001”, the element `ebics/AuthSignature/ds:SignatureValue` contains the result of the following operations for the identification and authentication signature:

1. Canonisation of the `ebics/AuthSignature/ds:SignedInfo` structure in accordance with “Canonical XML”
2. Hash value configuration via the canonised data with SHA-1
3. Signature computation via the calculated hash value with RSA: Here, the private identification and authentication key of the subscriber or – where available – the technical user is used in the case of EBICS requests, and in the case of EBICS responses the private identification and authentication key of the financial institution is used
4. base64-encoding of the signature.

## EBICS specification

The optional elements `ds:KeyInfo` and `ds:Object` in `ebics/AuthSignature` are not filled out:

- `ds:KeyInfo` can accept the public keys or certificates required for verification of an XML signature. With EBICS, these are already distributed and persistently stored at both the server's and client's end for key management order types "INI", "HIA", "HSA" and "HPB", and also additional order types for key alteration PUB and HCA
- `ds:Object` can contain a data object that is to be signed. However, within the framework of the EBICS identification and authentication signature, only those XML structured including sub-structures are signed that contain the attribute `@authenticate='true'` (namely the technical and order-related control data incl. preliminary checking data and transaction key information, in addition to the ES(s)).

The following shows an example of an EBICS message with identification and authentication signature in accordance with the XML signature:

```
<?xml version="1.0" encoding="UTF-8"?>
<ebics
  xmlns="http://www.ebics.org/H001"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ebics.org/H001 http://www.ebics.org/H001/ebics_request.xsd"
  Version="H001" Revision="1">
  <header authenticate="true">
  [...]
  </header>
  <AuthSignature>
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <ds:Reference URI="#xpointer(//*[@authenticate='true'])">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>9H/rQr2Axe9hYTV2n/tCp+3UIQQ=</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>Mx4psIy9/UY+u8QBJRDrwQWK...</ds:SignatureValue>
  </AuthSignature>
  <body>
  [...]
  </body>
</ebics>
```

For generation of the XML signature, see also:

<http://www.w3.org/TR/xmldsig-core/#sec-CoreGeneration>

For the canonisation algorithm "C14N", see also:

<http://www.w3.org/TR/2001/REC-xml-c14n-20010315>

### 4.4 Meaning of the XPointer expression in `ds:Reference@URI`

All elements of the EBICS message with the attribute `@authenticate='true'` and its sub-structures are referenced with the XPointer expression

"`#xpointer(//*[@authenticate='true'])`" for the XML signature attribute `ds:Reference@URI` (see <http://www.w3.org/TR/xmlsig-core/#sec-Same-Document>).

XPointer is a standard (see <http://www.w3.org/TR/WD-xptr>) that expands XPath notation (see <http://www.w3.org/TR/WD-xptr>) with additional constructs for localisation within XML documents. Here, the term "`#xpointer`" introduces labelling of the URI as an expression in accordance with the XPointer standard. The entry "`//*`" is a shorthand form that initially selects all elements of the XML document. "`[@authenticate='true']`" is used to restrict this selection to those elements that have an attribute `@authenticate` with the value "true". XML signature defines that when using XPointer syntax, the sub-structures of the elements selected in this way are included in the signature.

In contrast to explicit listing of all (sub-) structures that are to be identified and authenticated, the above expression has the following advantages in the EBICS context:

1. Succinct short form: Although a number of structures (both in the header data as well as the body data) are to be identified and authenticated, a short line is sufficient to specify the scope of the data that has been signed or that is to be signed. Sub-structures are automatically included in accordance with XML signature.
2. Immediate documentation of the scope of the signature: The schema itself immediately defines the elements and sub-structures that have been signed or that are to be signed via the attribute `@authenticate='true'`, whose appearance in the XML document is specified and documented directly in the XML schema definition. Therefore no separate concept document is required to check whether the specified data scope is covered by the signature; this task is already fulfilled in the schema validation.
3. Invariance in changes to the XML schema: Modifications to the XML structure (e.g. within the framework of a new EBICS version) do not influence the meaning of the `@authenticate` attribute values. Even if individual elements are re-named, removed, added or re-positioned, only the `@authenticate` attributes from the schema decide as to the scope of the data for signature configuration. On the other hand, if absolute paths were used in the URI reference specifications, in the event of any change the consistency of these would also have to be checked with regard to the changed XML structure.

### 4.5 Verification of the identification and authentication signature

The following steps are to be carried out for verification of the identification and authentication signature:

1. Validation of the EBICS message with regard to the matching EBICS schema. A positive result guarantees that the `@authenticate='true'` attributes are actually present at the places required in the schema.
2. Verification of the composition of the XML signature structure. Here, care should be taken that the reference URI and the canonisation and transformation algorithm corresponds with the specifications in Chapter 4.3 and that no additional transformation algorithms are entered. Deviating or additional transformation algorithms can have an influence on the type and scope of the data that is to be signed and can hence distort the results of the identification and authentication check.
3. Configuration of the hash value via the elements recorded by the reference URI in accordance with the specifications returned in `ebics/AuthSignature/ds:SignedInfo/ds:Reference` and comparison with the base64-decoded value of the returned element `ebics/AuthSignature/ds:SignedInfo/» ds:Reference/ds:DigestValue`. In the event of a difference between these values, the identification and authentication check is deemed to have failed.
4. Calculation of the signature hash value in accordance with the specifications in `ebics/AuthSignature/» ds:SignedInfo`, i.e. execution of operations 1 and 2 for filling out the element `ebics/AuthSignature/ds:SignatureValue` from Chapter 4.3. After successful checking of steps 1 to 1 of this verification procedure, the data from the XML signature structure is identical to the specifications of Chapter 4.3 for generating the identification and authentication signature.
5. Execution of RSA signature verification using the public RSA identification and authentication key of the other party on the base64-decoded version of the supplied signature value in the element `ebics/AuthSignature/ds:SignatureValue`. In the case of EBICS requests, the financial institution determines the identity of the other party via the control data (subscriber ID or technical system ID and customer ID), in the case of EBICS responses the customer system selects the appropriate key of the contacted financial institution. The result of the RSA operation is the signature hash value calculated by the other party.
6. Comparison of the two signature hash values. The identification and authentication signature is only successfully verified if the hash values are identical, i.e. the elements and sub-structures of the EBICS message with the attribute `@authenticate='true'` are authentic and the identity of the sender is assured.

For more details on the validation process of XML signatures, see:

<http://www.w3.org/TR/xmlsig-core/#sec-CoreValidation>

**EBICS specification**

EBICS – Implementation Guide Version 1.7

---

## 5 Replay avoidance using Nonce and Timestamp

### 5.1 Process description

In the case of EBICS messages that are to be executed over several steps, a transaction ID is supplied in the control data of the relevant EBICS message via which the message is unambiguously assigned to the current transaction. An exception to this basic principle is the EBICS request in the initialisation phase of the transaction, since the transaction ID is generated by the financial institution in the initialisation phase and only made known to the customer system in the first EBICS response.

The first EBICS request that serves for initialisation of an EBICS transaction therefore contains the elements **“Nonce”** and **“Timestamp”** that are together intended to prevent replay of this request:

- “Nonce“ is a cryptographically-strong random number with a length of 128 bits that is generated by the customer system (see also Chapter 7.1)
- “Timestamp“ contains the point in time (date and time in accordance with ISO 8601) at which the EBICS request was sent by the customer system.

“Nonce“ and “Timestamp“ form a functional unit for the avoidance of replay:

1. The customer system generates a random “Nonce“ and sets a “Timestamp“ at the current point in time that the message is sent.
2. The bank system compares the received “Nonce“ with a list of previously-received “Nonce“ values. In addition, it checks the deviation between the “Timestamp“ and the current time. If the “Nonce“ that has just been received is in the list, or if the deviation of the “Timestamp“ is greater than a fixed tolerance period (guideline value: a few hours plus/minus the bank system’s local time), the request is answered with the technical error return code “EBICS\_TX\_MESSAGE\_REPLAY“.
3. If the “Nonce“ and “Timestamp“ check was carried out without errors, the bank system stores the “Nonce“ and “Timestamp“ pair in the list and continues with the further processing of the message.

The bank system can delete “Nonce“/“Timestamp“ pairs whose time stamp lies outside the tolerance period from its list: Messages that contained such a pair would have already been rejected due to the excessive deviation of the “Timestamp“. Therefore the fixed tolerance period applies equally to the checking of new pairs as well as the deletion process of stored pairs.

With the elements “Nonce“ and “Timestamp“, this process guarantees that the contents of the first EBICS request of a transaction are new. This prevents the bank from initialising new EBICS transactions on the basis of old, replayed messages. At the same time, “Timestamp“ restricts the chronological necessity of the storage of “Nonce“ values by the bank.

However, “Nonce” and “Timestamp” are only designed to prevent “technical” replay attacks in the initialisation of EBICS transactions. “Business related” double submissions, such as identical submitted accounting figures on the same day, cannot be intercepted using this method. For this reason, business related control mechanisms are still required in the bank’s computing and accounting systems to check for double submissions e.g. of ES’s or order data.

## 5.2 Formats of “Nonce” and “Timestamp”

### 5.2.1 Format of “Nonce”

“Nonce” is to be fully filled out, including leading nulls if necessary, with a cryptographically-strong random number with a size of 128 bits in hexadecimal notation (32 digits from 0-9 and capital letters A-F in accordance with the canonical form of the XML schema type “hexBinary”). The selection of a random number of this size ensures that the probability of a conflict occurring between the “Nonce” values of transactions that are being executed in parallel (even those of other subscribers/customers) is sufficiently small (see Chapter 5.3.2).

### 5.2.2 Format of “Timestamp”

A time stamp in “ISO 8601” notation in accordance with the XML schema data type “dateTime” is to be used in a combined form comprising date and time for filling out the “Timestamp”. For the time zone, either UTC (Coordinated Universal Time, earlier called GMT) can be used (correspondingly, “Z” for “Zulu” is to be appended as a time zone marker) or the difference in comparison with UTC can be appended. In the latter case, aspects such as summertime must be taken into consideration.

In the EBICS context, the format of time stamps in accordance with ISO 8601 for combined specification of date / time is as follows:

*CCYY-MM-DDThh:mm:ss.tttZ* for UTC, or *CCYY-MM-DDThh:mm:ss.ttt±OO:oo* in the case of time specifications that deviate from UTC. Here, characters marked in bold are to be taken over unchanged (“±” means “either + or –”), the italic letters are wildcard characters:

- *CC* for the century,
- *YY* for the year of the century,
- *MM* for the month of the year (01 for January),
- *DD* for the day of the month,
- *hh* for the hour of the day (24-hour clock format, e.g. 15 for 3pm),
- *mm* for the minutes of the hour,
- *ss* for the seconds of the minute,
- *ttt* for the thousandths of the second,
- *OO* for the difference in hours to UTC,
- *oo* for the difference of the minute proportion to UTC,

### 5.3 Actions of the customer system

#### 5.3.1 Filling out the fields “Nonce” and “Timestamp”

For the initial standard EBICS request of a transaction, the customer system must fill out the elements “Nonce” and “Timestamp” in the technical header data in accordance with the format specification in Chapter 5.2, wherein the current time stamp of the customer system is to be used for “Timestamp”.

In the case of initial key management EBICS requests (i.e. only for order types “INI“, “HIA“, “HSA“, “HPB“ and “SPR“), the fields “Nonce” and “Timestamp” are only to be filled out if an identification and authentication signature is required in the request for the selected order type (i.e. with EBICS protocol version “H001“ only “HPB“ and “SPR“, see Chapter 4.1).

An example of syntactically-correct setting of the values “Nonce” and “Timestamp” is shown in the following XML excerpt:

```
<?xml version="1.0" encoding="UTF-8"?>
<ebics
  xmlns="http://www.ebics.org/H001"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ebics.org/H001 http://www.ebics.org/H001/ebics_request.xsd"
  Version="H001" Revision="1">
  <header authenticate="true">
    <Nonce>01A56FF768B3B36C5120E9904A7FB035</Nonce>
    <Timestamp>2005-06-22T17:07:34.123+02:00</Timestamp>
    [...]
  </header>
  [...]
</ebics>
```

Further information on correct setting of the two XML schema elements can be found under <http://www.w3.org/TR/xmlschema-2/#hexBinary> (hexBinary) and <http://www.w3.org/TR/xmlschema-2/#dateTime> (dateTime).

#### 5.3.2 Behaviour in the event of error message „EBICS\_TX\_MESSAGE\_REPLAY“

The bank system uses the technical error code EBICS\_TX\_MESSAGE\_REPLAY to signal that the EBICS message that has just been sent by the customer system contains a “Nonce” value that corresponds with one stored in the bank system, or that the “Timestamp” lies outside the tolerance period.

The use of cryptographically-strong random numbers as “Nonce” practically excludes the coincidental occurrence of such a situation. With the specified Nonce length of 128 bits, the probability of any conflict between two independently-generated random Nonces is precisely  $2^{-64}$ , that is, on average a conflict of this nature will occur once in approx.  $1.845 \cdot 10^{19}$  cases. In addition, this conflict would have to occur within the tolerance period, usually a few hours.

Therefore after receipt of the report EBICS\_TX\_MESSAGE\_REPLAY, the customer system must take into account the possibility of a replay attack, an intolerably-imprecise clock setting in the customer system or the bank system, or an error in its own transaction management in the assignment of “Nonce” values.

If the subscriber would nevertheless like to successfully transmit the EBICS message in question, they must at least first regenerate the fields “Nonce” and “Timestamp” in accordance with Chapter 5.2. In addition, the identification and authentication signature must be regenerated since it also includes the two fields “Nonce” and “Timestamp” in the signature.

## 5.4 Actions of the bank system

### 5.4.1 Checking “Nonce” and “Timestamp”

When the bank system receives an initial EBICS message from a subscriber, it must carry out the following actions to check for message replay.

1. Validation of the formats of the received technical EBICS header data “Nonce” and “Timestamp” within the framework of schema validation. For standard EBICS requests, validation is carried out against the schema “ebics\_request.xsd”. Validation is carried out against the schema “ebics\_keymgmt:request.xsd” in the case of key management EBICS requests. In the event of a key management request, it should be additionally noted that specification and filling out of the two fields is imperative when an identification and authentication signature is also a prerequisite for the given order type (i.e. with H001, only for order type “SPR”, see Chapter 4.1).

If the schema check proves to be negative, the message has not been constructed in accordance with EBICS guidelines. Therefore the bank system must reply with the technical error code “EBICS\_INVALID\_REQUEST”.

If a request has been received within the framework of key management for which a “Nonce” / “Timestamp” entry is not required, the replay check is dispensed with.

2. Comparison between the received “Timestamp” and the local time stamp: Normalised to UTC, the received “Timestamp” must be within the tolerance period that is stretched around the current time stamp of the bank system. This tolerance period will compensate for differences in precision between the clocks involved in the systems and possibly also early/late changeover to summer/wintertime. At the same time, the tolerance period determines when the bank system can delete stored “Nonce”/“Timestamp” pairs. An incoming message with a “Timestamp” outside the tolerance period would not be accepted, therefore the stored “Nonce”/“Timestamp” pairs with a “Timestamp” outside the tolerance period are superfluous and can be deleted.  
The tolerance period must be set as a one-off occurrence by the bank system. The setting “± 6 hours” can serve as a concrete guideline value. Higher values (= large tolerance periods) increase the storage requirements for valid “Nonce”/“Timestamp”

pairs, lower values (= smaller tolerance periods) increase the risk of rejected EBICS messages as a result of excessive clock differences between customer & bank systems.

If the received “Timestamp” is not within the tolerance period there is a risk of message replay. Therefore the bank system must reply with the technical error code “EBICS\_TX\_MESSAGE\_REPLAY”.

3. Comparison of the received “Nonce” with the “Nonce” values stored in the bank system. All “Nonce”/“Timestamp” pairs that originate from valid EBICS requests within the tolerance period are stored at the bank’s end. If the received “Nonce” corresponds with a stored “Nonce” the bank system must reply with the technical error code “EBICS\_TX\_MESSAGE\_REPLAY”.

If the above checks are all passed, there is no message replay. The bank system can then proceed with the actions in Chapter 5.4.2.

### 5.4.2 Storage of “Nonce”/“Timestamp” pairs

If the received “Nonce”/“Timestamp” pair is successfully checked, the bank system must store this pair for later checks on other EBICS messages. If a database is used for this purpose, the “Nonce” field can be used as an indexed primary key of a database table, which has a number of advantages:

- “Nonce is a non-empty alphanumeric value of fixed length (representation as “hexBinary” corresponds to a string of length 32, formed from the characters “0”-“9” and “A”-“F”) that unambiguously labels the message. Hence it is suitable for use as a primary key.
- Later access in the framework of conflict checks always takes place on the basis of the similarity check. Therefore an index to this field means a substantial speed increase for such tests.

Other primary keys are not provided, i.e. storage of the “Nonce”/“Timestamp” pairs takes place throughout the bank system.

The “Timestamp” field is required to restrict the validity of the “Nonce” field. Range comparisons are used for this task (e.g. “Timestamp” ≤ “current timestamp” + 6 hours and “Timestamp” ≥ “current timestamp” - 6 hours).

A deletion process that thins the expired “Nonce”/“Timestamp” pairs at regular intervals (guideline value: once per hour) can remove precisely those entries whose “Timestamp” entry is no longer within the tolerance period. In this way, the size of the table remains limited to the maximum number of valid EBICS requests within the tolerance period.

The guideline value for the deletion interval is a compromise between the utilisation of resources with regard to computing time and storage space. The more frequently the deletion process is active, the more computing time will have to be expended in searching the database, and the fewer expired entries will accumulate in the table in the meantime. If the time period is too short, the deletion process will only find a few expired entries, which means that the search would hardly be worth carrying out. On the other hand, if the time limit

between two deletion processes is too long each replay check for double “Nonces” will also have to cope with a large number of expired entries.

The bank system’s protection from manipulation of the data in connection with message replay is decisive for the validity of the above EBICS check. To achieve this, the following data and system components must imperatively be secured against compromise at the bank system’s end:

- The stored “Nonce”/“Timestamp” pairs
- The “Nonce”/“Timestamp” pairs of the EBICS request message that is to be checked
- The bank system’s internal clock
- The deletion process instructions.

### 5.4.3 Further Recommendations

#### **DoS attacks by registered users/clients**

At present, there is no limitation on a customer's number of sessions. Thus, a single customer may (accidentally) open numerous sessions. With version 2.1 of the EBICS detailed concept a corresponding error code has been introduced for banking systems wishing to impose a limitation on https sessions (09-1-1-19 (EBICS\_MAX-TRANSACTIONS\_EXCEEDED)). A maximum number of parallel https sessions for each customer ID can be specified. The maximum number of sessions per customer can be parameterized.

## 6 XML parsers and generators

### 6.1 Tasks

An XML parser has the following fundamental tasks:

- Verification of the syntactic correctness (“well-formedness”) of the XML document that is to be analysed
- Validation of the XML document against a specified schema (“validity”)
- Provision of the data present in the XML document via a programming interface (API)

On the other hand, an XML generator is intended for the following range of tasks:

- Provision of a programming interface (API) for the transfer of data for an XML document that is to be generated
- Ensuring the syntactic correctness (“well-formedness”) of the generated XML document
- optional: Validation of the generated XML document against a specified schema (“validity”).

Validation against an existing schema is optional for an XML generator since the application that generates the XML document must already do this by knowing which schema to use. Hence validation is only an additional security step.

### 6.2 Models

The similarity of the range of tasks for XML parsers and XML generators (see Chapter 6.1) means that in many cases, an XML parser is suitable for the generation of XML and vice-versa.

With XML generators and parsers, a differentiation is drawn between two processing models:

1. **tree-based:** With this model, the XML is completely displayed as a tree structure in the main memory. With the help of API, the user can electively and even multiply run through the tree structure, add or delete new branches and pages and can effect generation at any time of the XML document that is represented as a tree structure & that is syntactically correct.  
This group includes the so-called “DOM parsers“ (Document Object Model).
2. **event-driven:** This model uses so-called “event handlers“ that sequentially process an XML document as a data stream (“streaming”). If such a parser discovers an individual data object in the XML stream, a specific event is triggered and transmitted to the application via the API. The information relating to all data of the object (name, type, etc.) is linked to the event. The application can store this information and then initiate further processing via the API, i.e. trigger the next event. Conversely, when generating XML documents in accordance with this model the events are triggered by the application itself via the API and, together with the

transmitted information, thereby precipitate generation of the appropriate data object in the previously-initialised data stream.

This group includes the so-called “SAX parsers” (Simple API for XML).

“DOM parsers” (<http://www.w3.org/DOM/>) and “SAX parsers” (<http://www.saxproject.org/>) are widespread representatives of their respective type and both support the generation of new XML documents as well as the analysis of existing XML documents. For this reason, the term “DOM parser” will be used to denote tree-based XML parsers and generators, and similarly “SAX parser” will be used to denote event-driven XML parsers and generators.

The two approaches have the following characteristics:

- DOM parsers always keep the entire XML document in memory as a tree structure, i.e. it uses a “view” of the whole document. In this way, memory consumption increases in proportion with the size of the XML document. Moreover, the application is not forced to intermediately store further XML-specific data. DOM parsers ensure that the structure is syntactically correct at all times. The application can gain access electively, i.e. can freely jump to any position in the tree (even multiple times) where it can read, change or add structures. Checking the validity of an existing XML document takes place by completely importing the XML schema followed by validation against the XML tree structure representation.
- SAX parsers analyse an XML document sequentially at the data object level, i.e. in each case they “see” an atomic section of the XML document. Memory consumption is therefore constant and independent of the size of the XML document. For this, the application must store and continually adapt the current context that is defined by the events, i.e. depending on semantics and the purpose of the analysis, additional buffer storage will be required for XML-specific data at the application level. A SAX parser does not diagnose a syntactic defect until it reaches the corresponding point in the XML document. For validation via XML schema, the parser first runs through both the schema and the XML file once completely, and signals errors as usual by triggering events that contain additional information such as line, column and the name of the defective XML object.

In addition to these two “pure forms”, there are other approaches that attempt to combine the positive characteristics of the respective other model with their own. In this way, the framework “SAXDOMIX” (<http://www.devsphere.com/xml/saxdomix/>) is based on the SAX parser but the application can create a DOM sub-tree for the current element at any time. In contrast, “XML Pull Parsers” (XPP: <http://www.xmlpull.org/>, <http://www.extreme.indiana.edu/xgws/xsoap/xpp/> and <http://www.extreme.indiana.edu/xgws/xsoap/xpp/mxp1/>) are not based on events that are triggered by the parser, instead the application itself provides the commands to parse. The API is rather similar to the DOM, with the difference that like the SAX, only an atomic section of the XML document is present in memory at a given time.

### 6.3 Selection of a suitable model

In general, both DOM and SAX parsers can cover all tasks that are to be expected in the EBICS context.

The following list of arguments may be useful in selecting the right parser for the implementation process:

- DOM parsers are more suitable for documents that fit completely into the available main memory. SAX parsers are particularly expedient when it cannot be ensured that the supplied XML document will fit completely into the main memory.
- The “mongrel versions“ such as “SAXDOMIX” or “XPP” are interesting alternatives but an additional library will definitely be required for your use (SAX and DOM, on the other hand, are contained as standard in the more up-to-date Java SDKs).
- With SAX, the application must take on more tasks for administration of the information and the context that is the case with DOM. Therefore with clever programming, SAX implementations can parse an XML document much faster and more efficiently than comparable DOM implementations.

With EBICS, the following special aspects must be taken into consideration:

- In the case of protocol version “H001“, within the framework of segmentation the order data attains a size of 1 MB, the remaining data of the EBICS message must be added to this. If a number of such requests arrive almost simultaneously in the bank system, DOM parsers, due to their characteristic of being able to hold the complete XML structure in the main memory, could reach the boundaries of available memory if load distribution does not take place beforehand. On the other hand, with SAX parsers the size of the XML message is not important as regards main memory utilisation, i.e. in the case of increasing parallelisation their main memory load still remain lower than with DOM parsers.
- The elements accessed by the identification and authentication signature via `@authenticate` are distributed over the entire XML document. The identification and authentication signature itself is between the header and the body data.
  - A SAX parser can use its event mechanism to verify the identification and authentication signature. So long as the attribute value `@authenticate="true"` appears in the data stream, the associated element and the sub-structure can be buffered for further processing. A SAX parser must also buffer this for comparison with the supplied identification and authentication signature.
  - After parsing the XML message, a DOM parser must run through the internally-configured XML tree again completely to localise all `@authenticate` attributes.

## 7 Random numbers

### 7.1 Cryptographically-strong pseudo-random numbers

**Cryptographic** (or cryptographically-secure) **pseudo-random number generators** (PRNG) are deterministic algorithms via which a sequence of **non-predictable** random numbers can be generated based on a real random number as a starting value (seed). In this context, non-predictable means that no further random numbers that have already been generated by the PRNG in the past or that will be generated in the future can be derived from knowledge of a few already-generated random numbers.

**Cryptographically-strong** (or cryptographically-secure) **pseudo-random numbers** are generated by cryptographic PRNGs.

With EBICS, cryptographically-strong random numbers are used in transaction management and in the avoidance of replay: both transaction IDs and Nonces are cryptographically-strong random numbers with a length of 128 bits. The entropy of the utilised seed should be at least 100 bits. See also the regular publications of the “Overview on suitable algorithms” from the Regulatory Authorities for Telecommunications and Posts.

Examples of cryptographic PRNGs are:

- **PSGR based on block ciphers::**  
Here, block ciphers such as AES or even 3DES are used in counter mode, wherein the seed serves as a symmetric key
- **PSGR based on hash functions::**  
HMACs are used in counter mode, wherein the seed is the secret. For example, SHA-1 can be used as the hash function.

Selection of the seed is decisive for the security of the cryptographic PRNGs.

On Unix systems, `/dev/random` normally supplies real random number values, on the other hand `/dev/urandom` only supplies cryptographically-strong random values.

On Windows systems, cryptographically-strong random numbers can be generated with the help of the CryptoAPI function `CryptGenRandom( )`. On the other hand, Windows does not provide a function that supplies real random values.

General recommendations for the generation of the random seed, also using computer hardware, can be found in RFC 1750 (Randomness Recommendations for Security). In particular, examples of non-secure PRNGs are also given here.

### **7.2 Java class SecureRandom**

A component of Java JCA (Java Cryptography Architecture) API is the class `java.security.SecureRandom`, which provides the functionality of a cryptographic PRNG. The quality of the PRNG is dependent on the implementation of the respectively-used CSP (Cryptographic Service Provider).

The JCA service provider "SUN" contains the implementation of a cryptographic PRNG that is based on the proprietary algorithm "SHA1PRNG" and corresponds with the standard IEEE P1363 (Appendix G.7).

## **8 Transaction management**

### **8.1 Transaction management at the customer's/subscriber's end**

The main responsibility of the transaction administration at the customer's end is control of the flow of EBICS transactions by initiation of the individual transaction steps. In doing this, the control of the flow must implement the stipulations from the EBICS detailed concept. Furthermore, transaction administration at the customer's end is responsible for the assignment of a transaction ID (or an EBICS message that contains this transaction ID) to the corresponding bank system.

The waiting time for an EBICS response should be restricted at the customer's end. The maximum waiting time should be a configurable magnitude whose value should be a matter of minutes, dependent on the quality of the Internet connection of the customer system. After expiry of this maximum waiting time, the corresponding transaction step should be deemed to have failed. After this, the client can use the recovery mechanism for EBICS transactions insofar as recovery is supported by the bank system in question.

At the customer's end, a counter should check the number of recovery attempts for any number of transaction steps. If the counter exceeds a fixed maximum value, the entire transaction should be deemed to have failed.

#### **8.1.1 Replay avoidance**

For the avoidance of replay in an EBICS request, so-called "Nonces" are transmitted in the first step of an EBICS transaction. This prevents the possibility of the same message being replayed unnoticed, resulting in the generation of an EBICS transaction.

Like transaction IDs, Nonces are implemented as cryptographically-strong random numbers with a length of 128 bits. See Chapter 7.1 for details on generation.

### **8.2 Transaction management at the bank's end**

#### **8.2.1 Lifespan of an EBICS transaction**

At the bank's end, the transaction management is responsible for the generation, the cancellation and the regular completion of EBICS transactions.

Open EBICS transactions are abandoned by customer systems if e.g. the Internet connection is interrupted for a long time period. It is also conceivable that open transactions cannot be proceeded due to software crashes at the customer's end.

The bank system's transaction management should terminate open transactions that are not closed by the corresponding subscriber (or technical subscriber). This can be attained e.g. by the stipulation of a maximum time period of inactivity in which the state of the transaction

does not change. The transaction would be terminated if such a time period is exceeded. The maximum period of inactivity should be a configurable magnitude. The actual value depends on how long the bank system wishes to give subscriber to continue an open transaction.

### 8.2.2 Transaction states

The next two sub-chapters contain state transition diagrams for upload and download transactions. The following variables/parameters will be used:

- **RecoveryCounter** contains the number of recovery attempts that have already taken place for the current upload transaction. Within the framework of transaction initialisation, RecoveryCounter is initialised with 0.
- **MaxRecovery** denotes the maximum permitted number of recovery attempts for EBICS transactions. MaxRecovery is equal to 0 if the bank system does not support recovery.
- **MaxIdleTime** is the maximum time period of inactivity for EBICS transactions.
- **TotalNr** denotes the total number of order data segments that are to be transmitted within the framework of the current upload transaction

Comments on notation:

The state transitions are labelled by expressions in the following format:

[<Event>][ “[<Condition>,” ] [“<Action>”]

Example:

Receipt of data segment n [ RecoveryCounter < MaxRecovery ] / RecoveryCounter ++

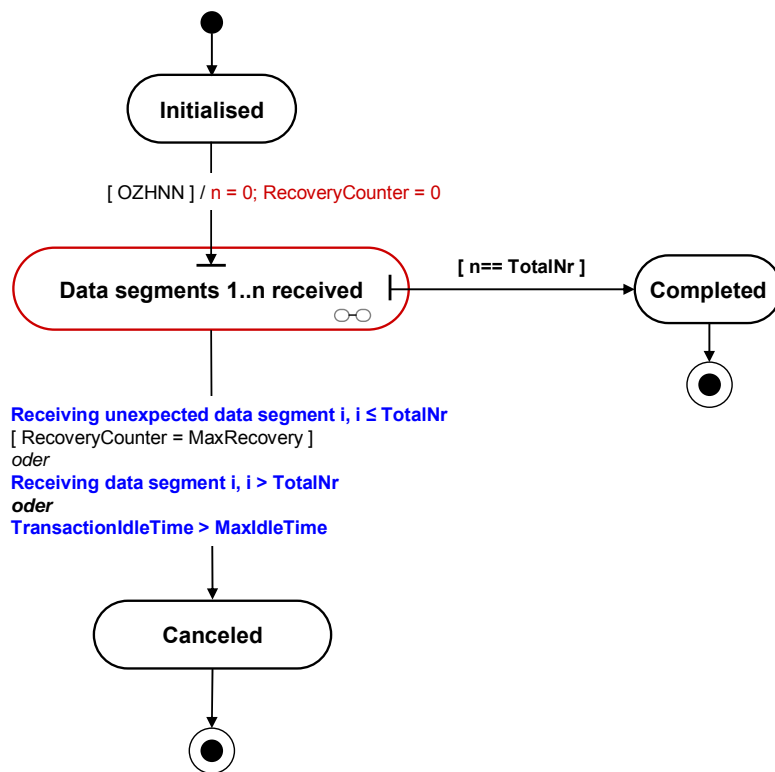
The meaning of such an expression is as follows:

- The state transition can be triggered by a specific event.  
e.g. Receipt of data segment n.
- In addition to the occurrence of an event, the state transition can be coupled with the fulfilment of a condition.  
e.g. RecoveryCounter < MaxRecovery
- The state transition can trigger the execution of an action.  
e.g. RecoveryCounter ++

The following state transition diagrams contain structured states. A state transition that is based on a structured state (and not on states within the structured state) is a simplifying equivalent notation for a quantity of state transitions with the same labelling that are based on each individual state within the structured state.

#### 8.2.2.1 State diagram of an upload transaction

The state transition diagram of an upload transaction is shown by way of example in Diagram 9 and Diagram 10.

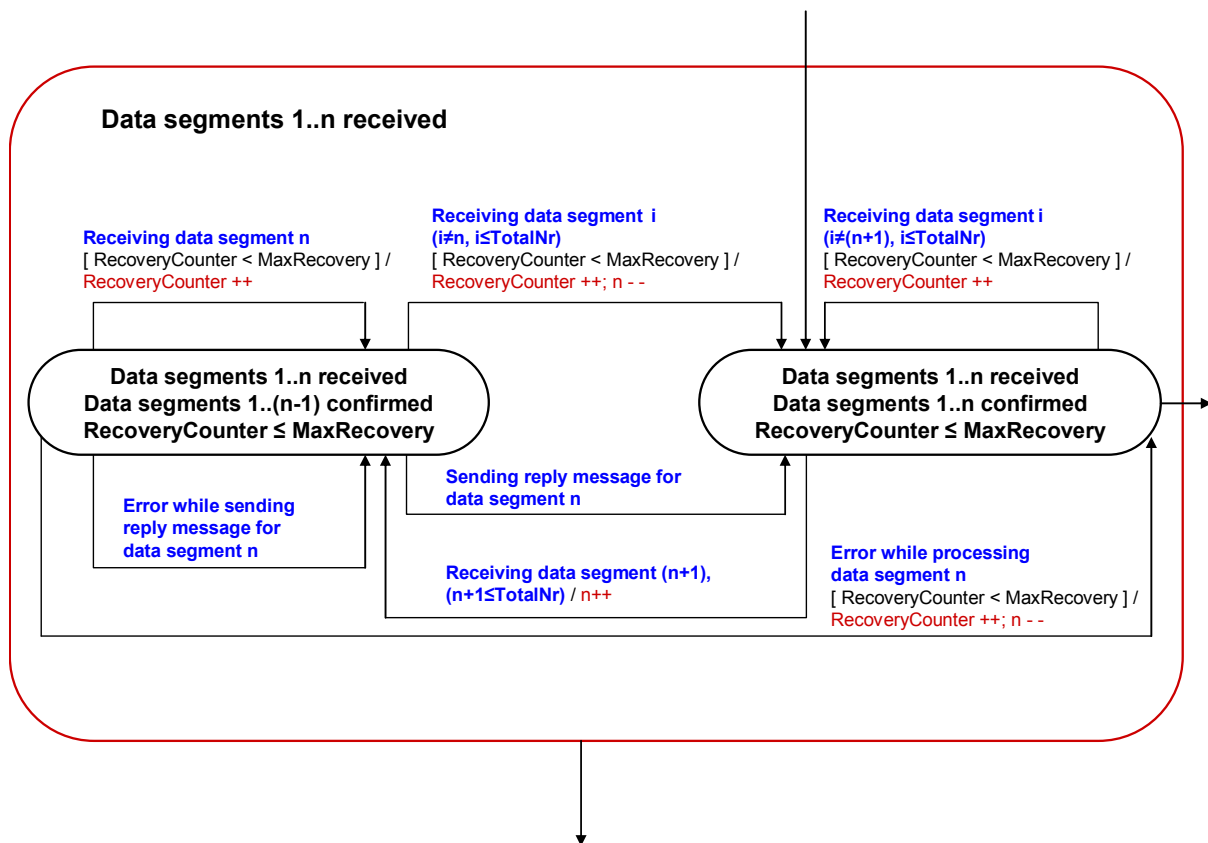


**Diagram 9: State transition diagram for upload transactions**

The state “initialised” is the state of the upload transaction after transaction initialisation has been successfully executed. The state “data segments 1..n received” combines the states of the upload transaction that these take after receipt of the  $n^{\text{th}}$  data segment and before receipt of the  $(n+1)^{\text{th}}$  data segment. After successful receipt of the last data segment the EBICS transaction is finally ended. The following events lead to cancellation of the upload transaction by the bank system:

- Exceeding the MaxIdleTime
- Receipt of an order data segment with a serial number greater than TotalNr
- Receipt of an unexpected order data segment when recovery of the transaction is not, or is no longer, possible

Recovery of the transaction is not possible when the bank system fundamentally does not support recovery or when the number of previous recovery attempts has reached the threshold specified by MaxRecovery.



**Diagram 10: Detailed representation of the structured state “Waiting for data segment”**

The structured state “data segments 1..n received” combines the following (partial) states:

- **Data segment 1..n received, data segment 1..n confirmed, RecoveryCounter ≤ MaxRecovery**

An EBICS transaction is in this state when the transaction step of the phase data transfer to transmission of the  $n^{\text{th}}$  data segment has been successfully ended and the following transaction step for transmission of the  $(n+1)^{\text{th}}$  data segment has not yet been initiated. Hence processing of the  $n^{\text{th}}$  data segment of the EBICS request was successful and no errors occurred on sending the corresponding EBICS response. In this state, the transmission of the  $(n+1)^{\text{th}}$  data segment is expected.

If a data segment with a serial number greater than TotalNr is transmitted whilst in this state, the upload transaction is terminated. (See transition of “Data segment 1..n received” after “Cancelled” in Diagram 9.)

The receipt of a data segment with a serial number that is not equal to  $(n+1)$  causes the transaction to be terminated if the bank system does not support recovery or if the RecoveryCounter has already reached the permissible maximum value MaxRecovery. (See transition of “Data segment 1..n received” after “Cancelled” in Diagram 9.)

Otherwise synchronisation must be carried out between the customer system and the bank system. This is achieved via the technical return code EBICS\_RECOVERY\_SYNC and transmission of the recovery point (here: transmission of data segment  $n$ ) in the EBICS response. In this case, only the RecoveryCounter changes in the state of the transaction – it is increased by 1. (See transition with the labelling “Receipt of data segment  $i$  ( $i \neq (n+1)$ ,  $i \leq \text{TotalNr}$ ) [ RecoveryCounter < MaxRecovery ] / RecoveryCounter ++;” in Diagram 10.)

After receipt of the expected data segment with serial number (n+1) within an EBICS request, the transaction shifts into the partial state in which sending of the corresponding EBICS response is still outstanding. (See transition with the labelling “Receipt of data segment (n+1), (n+1≤TotalNr) / n++” in Diagram 10.)

In the special case “n=0”, this state actually means “No order data received yet”. Upload transactions attain this state via a direct state transition from the state “Initialised” when not only ES’s but also order data is to be transmitted within the framework of this upload transaction.

- **Data segment 1..n received, data segment 1..(n-1) confirmed, RecoveryCounter ≤ MaxRecovery**

An upload transaction is in this state when one of the following conditions is met:

- Processing of the EBICS request with the n<sup>th</sup> data segment has not yet been completed
- Processing of the n<sup>th</sup> data segment has been successfully completed but the EBICS response has not yet been sent
- Processing of the n<sup>th</sup> data segment has been successfully completed but errors occurred during transmission of the EBICS response

In this state, re-transmission of the n<sup>th</sup> data segment can be effected by the subscriber if the response time of the bank system exceeds the timeout at the customer’s end and the bank system supports recovery.

In the event of re-transmission of the n<sup>th</sup> data segment the transaction is terminated if the bank system does not support recovery or if RecoveryCounter has already reached the permitted maximum value MaxRecovery. (See transition of “Data segment 1..n received” after “Cancelled” in Diagram 9). Otherwise only the RecoveryCounter changes in the state of the transaction – it is increased by 1. Additional synchronisation between the subscriber and the bank system is not necessary since the subscriber has selected the correct recovery point. In this event, the technical return code is EBICS\_OK and not EBICS\_RECOVERY\_SYNC. (See transition with the labelling “Receipt of data segment n [ RecoveryCounter < MaxRecovery ] / RecoveryCounter ++”.)

In this state, re-transmission of another segment other than the n<sup>th</sup> data segment can also be effected by the subscriber, such as in the case of loss of the transaction state at the subscriber’s end.

If a data segment with a serial number greater than TotalNr is transmitted whilst in this state, the upload transaction is terminated. (See transition of “Data segment 1..n received” and “Cancelled” in Diagram 9).

The receipt of a data segment with a serial number that is not equal to n causes the transaction to be terminated if the bank system does not support recovery or if the RecoveryCounter has already reached the permissible maximum value MaxRecovery. (See transition of “Data segment 1..n received” after “Cancelled” in Diagram 9).

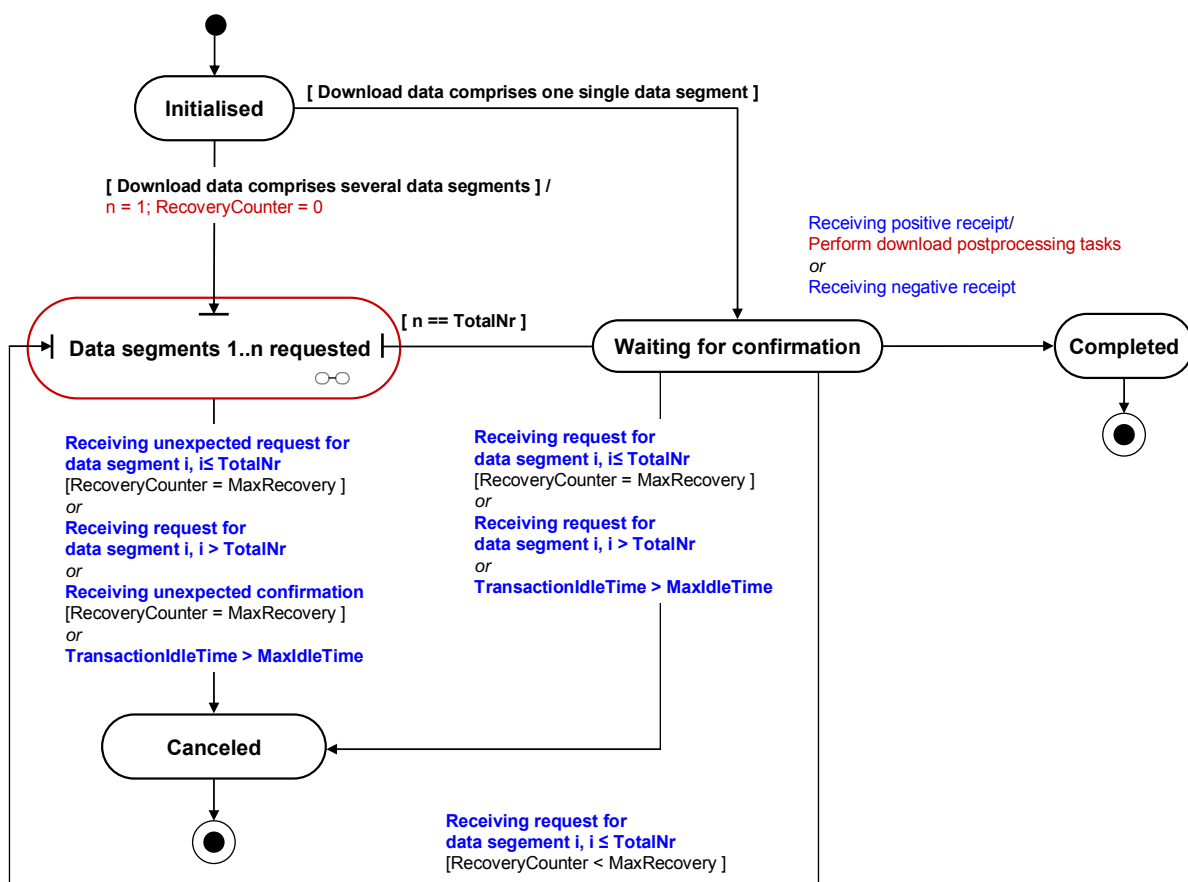
Otherwise synchronisation must be carried out between the customer system and the bank system. This is achieved via the technical return code EBICS\_RECOVERY\_SYNC and transmission of the recovery point in the EBICS response. The recovery point is the transaction initialisation where n=1 or the transmission of the (n - 1)<sup>th</sup> data segment where n>1. The state of the transaction is returned to the point at which the transaction

step of the recovery point had been successfully completed. In addition, the RecoveryCounter is incremented by 1. (See transition with the labelling “Receipt of data segment  $i$  ( $i \neq n, i \leq \text{TotalNr}$ ) [ RecoveryCounter < MaxRecovery ] / RecoveryCounter ++;  $n - -$ ” in Diagram 10.)

If errors in the bank’s processing of the  $n^{\text{th}}$  data segment necessitate re-transmission of this data segment, this is effected by synchronisation between the subscriber and the bank system. In this case, the technical return code is EBICS\_RECOVERY\_SYNC. The recovery point is the transaction initialisation where  $n > 1$  or the transmission of the data segment ( $n - 1$ ) where  $n > 1$ . The state of the transaction is returned to the point at which the transaction step of the recovery point had been successfully completed. The recovery counter is incremented by 1. (See transition with the labelling “Error in processing data segment  $n$  [ RecoveryCounter < MaxRecovery ] / RecoveryCounter ++;  $n - -$ ”.)

**8.2.2.2 State diagram of a download transaction**

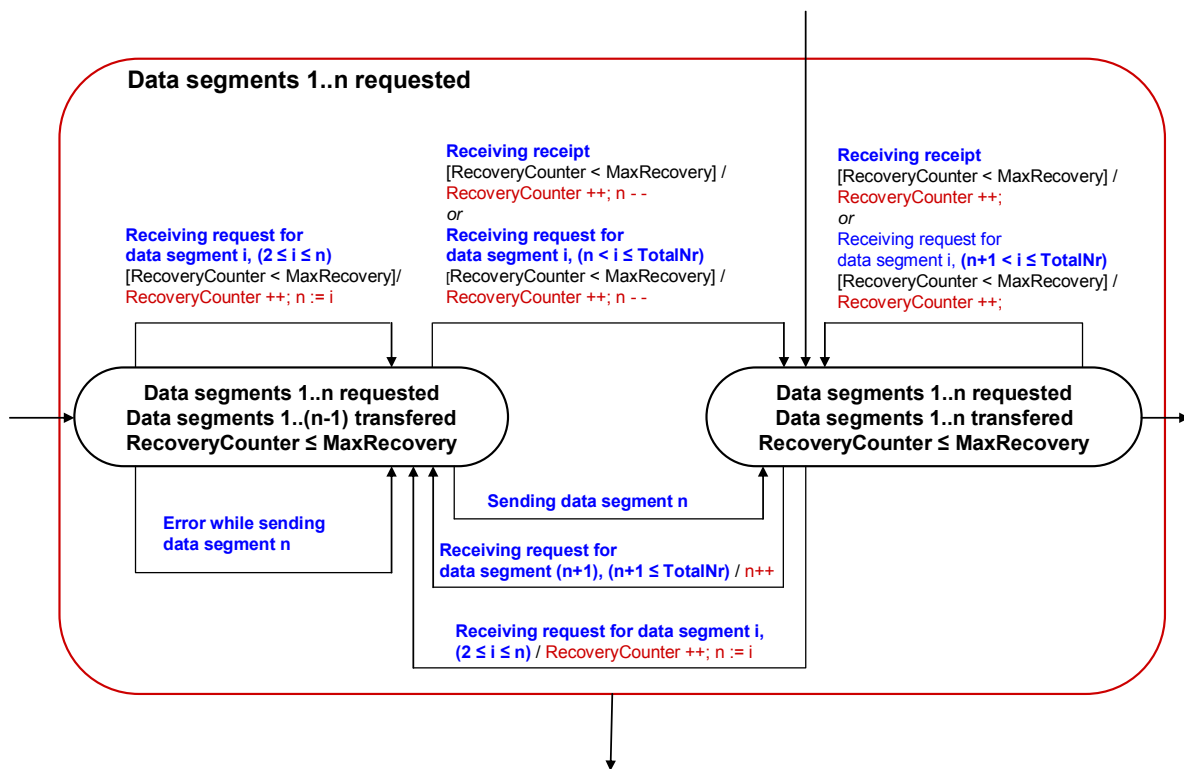
The state transition diagram of a download transaction is shown by way of example in Diagram 11 and Diagram 12.



**Diagram 11: State transition diagram for download transactions**

The state “initialised” is the state of the download transaction after transaction initialisation has been successfully executed (including transmission of the first order data segment). The state “data segments 1..n requested” combines the states of the download transaction that these take after requesting the  $n^{\text{th}}$  data segment and before requesting the  $(n+1)^{\text{th}}$  data segment. After successful transmission of the last order data segment the transaction waits for receipt of the acknowledgement (see the state “Waiting for acknowledgement”) before the download transaction can be completed. The following events lead to cancellation of the download transaction (see transitions of “Data segment 1..n requested” after “Cancelled” as well as “Waiting for acknowledgement” after “Cancelled” in Diagram 11).

- Exceeding the MaxIdleTime
- Requesting an order data segment with a serial number greater than TotalNr
- Receipt of the request for an order data segment that does not match the state of the transaction, when at the same time it is not, or it is no longer, possible to recover the transaction.  
This can be the request for an order data segment at a time when the previously-requested order data segment has not yet been transmitted. It can also be the request for an order data segment that is not in the correct sequence.  
Recovery of the transaction is not possible when the bank system fundamentally does not support recovery or when the number of previous recovery attempts has reached the threshold specified by MaxRecovery.
- Receipt of an acknowledgement when not all data segments have been transmitted and at the same time it is not, or it is no longer, possible to recover the transaction.  
Recovery of the transaction is not possible when the bank system fundamentally does not support recovery or when the number of previous recovery attempts has reached the threshold specified by MaxRecovery.



**Diagram 12: Detailed representation of the structured state “Waiting for request for a data segment”**

The state “Waiting for request for a data segment“ contains the following (partial) states:

- **Data segments 1..n requested, data segments 1..(n-1) transmitted, RecoveryCounter ≤ MaxRecovery**

An EBICS transaction is in this state when one of the following conditions is met:

- Processing of the EBICS request with the request for the  $n^{\text{th}}$  data segment is not yet completed, transmission of the EBICS response with the  $n^{\text{th}}$  data segment is still outstanding.
- Processing of the EBICS request with the request for the  $n^{\text{th}}$  data segment has been successfully completed but errors occurred during transmission of the EBICS response.

In this state, renewed request for the  $n^{\text{th}}$  data segment can be effected by the subscriber if, for example, the response time of the bank system exceeds the timeout at the customer’s end and the bank system supports recovery.

In the event of renewed request for the  $n^{\text{th}}$  data segment the transaction is terminated if the bank system does not support recovery or if RecoveryCounter has already reached the permitted maximum value MaxRecovery. (See transition of “Data segment 1..n requested” after “Cancelled” in Diagram 11).

## EBICS specification

Otherwise only the RecoveryCounter changes in the state of the transaction – it is increased by 1. However, additional synchronisation between the subscriber and the bank system is not necessary since the subscriber has selected the correct recovery point.

In this event, the technical return code is EBICS\_OK and not EBICS\_RECOVERY\_SYNC. (See special case  $i = n$  in the transition with the labelling “Receipt of request segm  $i$ , ( $2 \leq i \leq n$ ) [RecoveryCounter < MaxRecovery] / RecoveryCounter ++;  $n := i$ ” in Diagram 12.)

In this state, renewed request of the  $i^{\text{th}}$ , where  $i < n$ , data segments can also be effected by the subscriber, such as in the case of data loss at the subscriber’s end and simultaneous support of recovery by the bank.

The transaction is terminated in the event of renewed request for a data segment that has already been transmitted if the bank system does not support recovery or if the RecoveryCounter has already reached the permissible maximum value MaxRecovery. (See transition of “Data segment 1..n requested” after “Cancelled” in Diagram 11.)

Otherwise the download transaction is returned to the point where the request of the  $i^{\text{th}}$  data segment is expected. Also in this case the RecoveryCounter is incremented. Additional synchronisation between the subscriber and the bank system is not necessary since the subscriber has selected a permissible recovery point ( $i < n$ ). In this case, the technical return code is EBICS\_OK (and not EBICS\_RECOVERY\_SYNC), and the requested data segment is re-transmitted.

Synchronisation between the bank system and the subscriber is necessary when the subscriber requests a data segment  $i$ ,  $n < i \leq \text{TotalNr}$ , thereby jumping several data segments. When requesting such a data segment the transaction is terminated if the bank system does not support recovery, or if the RecoveryCounter has already reached the maximum permitted value MaxRecovery. (See transition of “Data segment 1..n requested” after “Cancelled” in Diagram 11.).

Otherwise the state of the download transaction is returned to the point where the request of the  $(n+1)^{\text{th}}$  data segment is expected. In this case, the technical return code is EBICS\_RECOVERY\_SYNC, the RecoveryCounter is incremented. The recovery point contained in the EBICS response is the transaction initialisation where  $n=2$  or the request for the  $(n - 1)^{\text{th}}$  data segment where  $n>2$ . (See transition with the labelling “Receipt of request segm.  $i$ , ( $n < i \leq \text{TotalNr}$ ) [RecoveryCounter < MaxRecovery] / RecoveryCounter ++;  $n --$ ” in Diagram 12.).

Synchronisation is also necessary between the bank system and the subscriber when the subscriber sends an acknowledgement without having requested all data segments. Acknowledgement of an order before receipt of the last data segment causes the transaction to be terminated if the bank system does not support recovery, or if the RecoveryCounter has already reached the maximum permitted value MaxRecovery. (See

transition of “Data segment 1..n requested“ after “Cancelled“ in Diagram 11). Otherwise the state of the download transaction is returned to the point where the request of the (n-1)<sup>th</sup> data segment is expected. In this case, the technical return code is EBICS\_RECOVERY\_SYNC, the recovery counter is incremented. The recovery point contained in the EBICS response is the transaction initialisation where n=2 or the request for the (n - 1)<sup>th</sup> data segment where n>2. (See transition with the labelling “Receipt of acknowledgement [RecoveryCounter < MaxRecovery] / RecoveryCounter ++; n --“ in Diagram 12.)

- **Data segments 1..n requested, data segments 1..n transmitted, RecoveryCounter ≤ MaxRecovery**

A download transaction is in this state when the transaction step of the phase data transfer for request of the n<sup>th</sup> data segment has been successfully ended and the following transaction step for request of the (n+1)<sup>th</sup> data segment has not yet been initiated.

In this state, the request for the (n+1)<sup>th</sup> data segment is expected. Receipt of the request for the (n+1)<sup>th</sup> data segment is the only event that triggers a transition outside of recovery. (See transaction with the labelling “Receipt of request segm (n+1), (n+1≤TotalNr) / n++“ in Diagram 6.)

All further state transitions take place analogously to the corresponding state transitions of the partial state “data segments 1..(n-1) transmitted, data segment n requested, RecoveryCounter ≤ MaxRecovery“ that has already been described.

## **9 Key management**

### **9.1 Generation of the RSA keys**

The algorithm for the generation of RSA keys is described in the „DFÜ-Abkommen“. See Appendix 2 (Security standards), Chapter 2.2.3.2: RSA key components. Further description and a detailed analysis of this algorithm is given in [2].

### **9.2 Generation of the symmetrical 3DES key**

Generation of the symmetrical key for the 2-key triple DES process is described in the „DFÜ-Abkommen“. See Appendix 2 (Security standards), Chapter 2.3.4.1: Processes at the sender's end.

### **9.3 Key storage**

This chapter considers alternatives for the storage of the private/public keys. Predominantly those alternatives are listed that are based on established standard formats and standard interfaces.

EBICS customer systems require access to the following keys:

- Subscriber's public and private keys
- Financial institutions' public keys via which communication is to take place using the EBICS protocol.

The bank systems require access to the following keys:

- Subscriber's public keys
- Private and public keys that are assigned to the bank system.

Storage of the keys must guarantee the integrity of the public keys and the integrity and secrecy of the private keys, both in the customer system and the bank system.

Components that are responsible for storage of the keys are generally referred to as keystores.

Keystores can be implemented as purely software components (software keystores). If different applications have to share a keystore, it is expedient to store the objects of the keystore such as keys, certificates, etc. directly in a standard format or at least to be able to export these in a standard exchange format.

---

<sup>2</sup> A. Menezes, P. van Oorschot and S. Vanstone, "Handbook of Applied Cryptography", Revised Reprint with Updates. CRC Press, 1997

As an alternative to software keystores, keystores can also be components of special crypto-hardware (Smartcards, HSM (Hardware Security Modules), USB tokens). In order to attain the independence of special hardware, it is important to implement access to the hardware via standard interfaces.

### 9.3.1 Standard formats

#### 9.3.1.1 PKCS#12

The standard PKCS#12 (Personal Information Exchange Syntax) v1.0 defines a secure exchange and storage format for both private keys and X.509 certificates. In this connection, secure means that both the integrity and the secrecy of the private key or the certificate is supported.

PKCS#12 ensures security of integrity and secrecy on the basis of passwords. A symmetrical key is derived from a (secret) password that is used for the encryption of data (e.g. the private key) whose secrecy is to be ensured. Analogously, a key is generated from a password that then flows into the calculation of the MACs (Message Authentication Code) of the data (such as e.g. certificates) whose integrity is to be ensured.

PKCS#12 is based on the standard PKCS#5 (Password-Based Encryption Standard) v2.0 which defines the process for the derivation of symmetrical keys from passwords. Furthermore, PKCS#12 is based on the standard PKCS#8 (the private-key information syntax standard) v1.2 which defines the format of secret keys.

PKCS#12 does not explicitly support the representation of public keys. Nevertheless, the standard allows password-based secure representation of any data types.

Popular browsers (Netscape, Mozilla, Firefox, Internet Explorer) use software keystores that store private keys and certificates in proprietary formats. However, the contents of these keystores can be exported in PKCS#12 format. Conversely, PKCS#12 files can be imported by the keystores of popular browsers.

Java keystores of type “pkcs12” support the PKCS#12 format (see also Chapter 9.3.2.3.1).

#### 9.3.1.2 XML

The following aspects of the XML standard can be used for the storage of public keys:

- The standard “XML signature” defines the (complex) XML type `KeyInfo` that allows the representation of public RSA keys as a combination of modulus and exponent (XML type `RSAPublicKey`) or as an X509 certificate (XML type `X509Certificate`).
- XML signature supports hash MACs as a signature algorithm, especially also HMAC SHA-1. These MACs can thus be used to ensure the integrity of XML elements on the basis of passwords.

On the other hand, there is no separate pre-defined XML type for representation of the private RSA keys.

EBICS provides for the transportation of public keys between the customer system and the bank system.

The order data for HIA, HSA, HPB or HCA orders are defined with the help of XML types `HIAResponseOrderData`, `HSARequestOrderData`, `HBPResponseOrderData` and `HCARequestOrderData`. In each case, these XML types contain elements of type `RSAPublicKey` or `X509Certificate`.

In contrast, the order data of INI or PUB orders have a proprietary binary format that conforms with the corresponding formats for remote data transmission via FTAM.

### 9.3.2 Standard interfaces for accessing keystores

The following APIs (Application Programming Interface) allow a uniform view of different crypto-hardware (Smartcards, HSMs (Hardware Security Module), USB tokens) from different manufacturers. The use of these APIs is not restricted to crypto-hardware, they can also be used for access to software keystores.

#### 9.3.2.1 PKCS#11 (Cryptographic Token Standard Interface)

PKCS#11 [11] is a C-API that has been developed by RSA Laboratories. The popular browsers Netscape, Mozilla and Firefox use PKCS#11 modules for accessing crypto-hardware.

#### 9.3.2.2 Microsoft CryptoAPI

Microsoft CryptoAPI is another C-API that is used exclusively by Windows applications.

In Windows, there is a central service that manages the CSP (Crypto-Service Provider) plug-ins for each computer. In addition, each user has a (software) keystore for the administration of user certificates whose (software) CSP is supplied with Windows. Access to the Windows keystores is via the CryptoAPI as standard, which is a component of the PC/SC (Personal Computer / Smartcard) interface.

#### 9.3.2.3 Java APIs

##### 9.3.2.3.1 Java JCA

JCA (Java Cryptography Architecture) is a plug-in based architecture and a framework of the Java 2 platform that is used to access cryptographic services. The functionality of such a service is defined via the interfaces of so-called engine classes, the concrete implementation of these interfaces is provided by means of CSPs (Cryptographic Service Providers).

A part of the JCA API is the class KeyStore from the package java.security, which is used for the administration of private keys and certificates. The main features of a Java keystore are:

- Password protection of individual sensitive entries (e.g. private keys) for the purpose of secrecy
- Password protection of the entire keystore for the purpose of verification of the integrity of the entire keystore.

There is a default CSP from Sun Microsystems called “SUN“, which is a component of the Java 2 SDK. Among other things, this contains the implementation of the Java keystore of type “jks“, which carries out file-based storage of the keys and certificates in proprietary “jks” format.

J2SE v1.5 contains a further CSP, the “Sun PKCS#11“ provider. This does not itself contain a PKCS#11 implementation but it allows the integration of PKCS#11 tokens in Java applications that are programmed against JCA APIs.

The Java 2 platform J2SE v1.4 contains the JSSE (Java Secure Socket Extension) provider “SunJSSE“, which among other things provides an implementation of the Java keystore type “pkcs12“. However, this implementation only supports read-only access to PKCS#12 files. In particular, “SunJSSE“ supports the use of pkcs#12 files that have been exported from the browser Netscape Navigator.

In J2SE v1.5, SunJSSE provides an improved version of the Java pkcs12 keystore that allows write access as well as read access.

### 9.3.2.3.2 Open Card Framework

OCF (Open Card Framework) is a Java framework for smart card applications that was developed with the intention of being independent of card manufacturer, card operating system and card terminal. OCF provides plug-ins for different cards and card terminals. Furthermore, OCF contains a card terminal adaptor for PC/SC.

## 9.3.3 Alternatives for key storage

### 9.3.3.1 Private keys

**Alternatives for storing the private subscriber keys at the customer system’s end:**

- Software keystores:
  - Java KeyStore (in the format JKS or pkcs12)
  - Microsoft KeyStore (for Microsoft CryptoAPI-based applications)
  - Software keystore (with application-specific interface) that stores the private keys in PKCS#12 or PKCS#8 format.
- Crypto-hardware:
  - Smart cards
  - USB tokens.

### Alternatives for storing the private bank keys at the bank system's end:

- Software keystores:
  - Software keystore that stores the private keys in PKCS#12 or PKCS#8 format.
- Crypto-hardware:

The private bank key should generally be stored using crypto-hardware.

  - HSM (Hardware Security Module).

### 9.3.3.2 Public keys

#### Alternatives at the customer's end:

- Storage of the public subscriber keys:
  - As a certificate (if applicable, self-signed) or in a proprietary format on the same crypto-hardware as the associated private key
  - As a certificate (if applicable, self-signed) in the Java KeyStore
  - In the case of CryptoAPI-based applications: As a certificate (if applicable, self-signed) in the Microsoft KeyStore
  - As an XML structure whose integrity is ensured via XML signature using HMACs. This XML structure is part of an application-specific software keystore
  - As a PKCS#12 structure that secures an XML structure or a proprietary binary format or a certificate (if applicable, self-signed)  
This PKCS#12 structure is part of an application-specific software keystore.
- Storage of the public bank keys:
  - As a certificate in the Java KeyStore, if the bank key is available as a certificate
  - In the case of CryptoAPI-based applications: As a certificate in the Microsoft Key Store, if the bank key is available as a certificate
  - As an XML structure whose integrity is ensured via XML signature using HMACs. This XML structure is part of an application-specific software keystore
  - As a PKCS#12 structure that secures an XML structure or a proprietary binary format or a certificate (if available).  
This PKCS#12 structure is part of an application-specific software keystore

#### Alternatives at the bank's end:

- Storage of the public subscriber keys:
  - As a certificate in the Java KeyStore, if the subscriber key is available as a certificate
  - In the case of CryptoAPI-based applications: As a certificate in the Microsoft Key Store, if the subscriber key is available as a certificate
  - As an XML structure whose integrity is ensured via XML signature using HMACs. This XML structure is part of an application-specific software keystore

- As a PKCS#12 structure that secures an XML structure or a proprietary binary format or a certificate (if available).  
This PKCS#12 structure is part of an application-specific software keystore
  
- Storage of the public bank key:
  - As a certificate (if applicable, self-signed) or in a proprietary format on the same crypto-hardware as the associated private key
  - As a certificate (if applicable, self-signed) in the Java KeyStore
  - In the case of CryptoAPI-based applications: As a certificate (if applicable, self-signed) in the Microsoft KeyStore
  - As an XML structure whose integrity is ensured via XML signature using HMACs.  
This XML structure is part of an application-specific software keystore
  - As a PKCS#12 structure that secures an XML structure or a proprietary binary format or a certificate (if applicable, self-signed).  
This PKCS#12 structure is part of an application-specific software keystore

### 9.4 Subscriber initialisation

In EBICS, subscriber initialisation comprises two separate orders: INI and HIA (exception: subscriber initialisation via HSA in the event of an existing FTAM key). The EBICS customer software should portray subscriber initialisation as a technical procedure and should conceal this division into two separate orders from the subscriber. Furthermore, the initialisation letters for INI and HIA should be sent together to the financial institution in order to simplify activation of the subscriber.

### 9.5 Verification of the bank keys

A prerequisite for the transmission of orders via EBICS is downloading of the bank key via EBICS using HPB, followed by verification of this bank key. For this reason, the financial institutions provide the bank keys via a second communication channel that is independent of EBICS. For example, the financial institution can publish the public keys together with their SHA-1 hash values via a portal of the financial institution. The hash value is formed by concatenating the exponent with a blank space and the modulus in hexadecimal representation without leading nulls. The hash values are composed by the financial institution as follows:

The hash values of the public X001 and E001 keys are composed by concatenating the exponent with a blank character and the modulus in hexadecimal representation (using lower case letters) without leading blanks (as to the hexadecimal representation). The resulting string has to be converted into a byte array based on US ASCII code.

The customer system must prompt the subscriber to verify the keys that are downloaded via HPB. The customer system must calculate the hash values of this key in order to allow comparison with the hash values published in the portal.

## **EBICS specification**

EBICS – Implementation Guide Version 1.7

---

If the hash value comparison is carried out manually by the subscriber, the associated subscriber must confirm the positive comparison.

**9.6 Amendment of the subscriber keys**

The EBICS customer software should portray the amendment of the subscriber key as a technical procedure. The division into individual orders HCA and PUB cannot be concealed from the subscriber since both orders require the subscriber’s deliberate signature. The following tables clarify which public/private subscriber keys are used, depending on the sequence, for processing HCA and PUB. Here, transmission without technical subscribers is considered.

			old subscriber key		new subscriber key	
			private	public	private	public
1.	HCA	Order data				<input checked="" type="checkbox"/>
		ES	<input checked="" type="checkbox"/>			
		Identification and authentication signature	<input checked="" type="checkbox"/>			
2.	PUB	Order data				<input checked="" type="checkbox"/>
		ES	<input checked="" type="checkbox"/>			
		Identification and authentication signature			<input checked="" type="checkbox"/>	

			old subscriber key		new subscriber key	
			private	public	private	public
1.	PUB	Order data				<input checked="" type="checkbox"/>
		ES	<input checked="" type="checkbox"/>			
		Identification and authentication signature	<input checked="" type="checkbox"/>			
2.	HCA	Order data				<input checked="" type="checkbox"/>
		ES			<input checked="" type="checkbox"/>	
		Identification and authentication signature	<input checked="" type="checkbox"/>			

If the transmission of the PUB and HCA order takes place via a technical subscriber, in each case the identification and authentication signature is formed with the identification and authentication signature of a technical subscriber but it may be advantageous to carry out HCA before PUB. This is particularly the case when e.g. the updating of the subscriber key is associated with the renewing of the subscriber’s chip card. Both orders are then signed with the subscriber’s old bank-technical key and are thus signed with the same chip card.

In general, the sequence of HCA and PUB should be selected so that the number of changes between the old and the new key that are visible to the subscriber are kept to a minimum.

### 10 Information on segmentation

In Version “H001” of EBICS, job data that requires more than 1 MB of storage space in compressed, encrypted and base64-coded form must be segmented before transmission, irrespective of the transfer direction (upload/download).

In the processes described in the EBICS detailed concept (Chapter 7), the following implementation-specific points are to be noted:

- As a result of the base64-encoding of the compressed, encrypted order data, the data volume increases in comparison with the unencoded form by a minimum of  $\frac{1}{3}$  and by a maximum of  $\frac{1}{3}$  plus 3 bytes.
- With EBICS, no characters are allowed within the base64-encoded data that are not part of the base64 alphabet (base64 alphabet upper-case letters “A”-“Z”, lower-case letters “a”-“z”, numbers “0” – “9”, plus sign “+”, oblique stroke “/”, equals sign “=”). For this reason, in deviation from the recommendations of RFC 2045, wherein after at most 76 characters of the base64 alphabet a carriage return / line feed (“CR/LF”) must be inserted, it is stipulated that all present “white space characters” (spaces, tabs, CR/LF etc.) are removed from the data block before separation into individual segments.
- Separation into segments each with a maximum of 1,048,576 bytes is to be effected in a manner that conforms with base64 so that the fields remain filled out in a valid fashion with regard to the EBICS schema (XML schema data type “base64Binary”). Concretely, the individual segments must have a byte size that is divisible by 4 .

As an alternative to immediate base64-encoding of the entire (compressed and encrypted) order data, it is also permissible to pre-calculate the data volume that results from the base64 encoding. Such a manner of proceeding allows the order data segments that have just been base64-encoded to be “streamed” directly into the individual EBICS messages without the sender having to pre-encode all segments before transmission. In this way, insofar as it is not necessary to re-synchronise via recovery during the transaction (see the EBICS detailed concept, Chapter 5.4), the sender of the order data saves local storage space without significant additional memory usage.

Even in the case of receipt, as an alternative to processing the data at the end of the transaction, each individual segment can be immediately base64-decoded and concatenated in binary form to the remaining data block, wherein the recipient can save  $\frac{1}{4}$  of the otherwise-used storage space.

The requirement of also formulating the individual segments in a base64-conformant manner allows uniform handling of the order data within the EBICS transaction independent of its quantity or the total scope of the order data.

Details on base64 can be found in RFCs 1421 (<http://www.ietf.org/rfc/rfc1421.txt>) and 2045 (<http://www.ietf.org/rfc/rfc2045.txt>).

## 11 Recommendations for filling out fields in the EBICS protocol

In the EBICS protocol, the contents of some fields are not restricted by the EBICS detailed concept. The following text contains recommendations for filling out these fields.

### 11.1 Manufacturer specifications for the customer product

The optional field `ebics/header/static/Product` SHOULD contain the designation of the customer product including the version and other specific characteristics (e.g. the "patch level" or the operating system for which the software has been developed). The product variant should be manufacturer-specifically identifiable with this designation, i.e. the manufacturer should be able to unambiguously assign to the product a specific point in their development history via the entry in this field.

The optional attribute `ebics/header/static/Product@InstituteID` SHOULD contain clear indication of the manufacturer that is unambiguous among customer products. Examples of such an indication are the company name as it appears in the commercial register or the URL of the manufacturer's website.

## **12 Further recommendations**

### **12.1 Actions of the customer system**

#### **Suspension request:**

After submitting a suspension the subscriber will not be able to effect any activity concerning remote data transmission any more. If the subscriber wants to conduct any finishing operations, e.g. cancellations, these have to be accomplished beforehand. The suspension can only be revoked by a reinitialisation. At the request of the customer, the bank may also revoke the suspension.

### **12.2 Actions of the bank system**

#### **Provision of end-of-record characters by the bank computer**

In the paragraph on general syntax rules for SWIFT formats in Appendix 3 of the DFÜ Agreement (see e.g. chapter 5 "Tageskontoauszugsinformationen") it is pointed out that end-of-record characters are to be provided as ASCII X'0D0A'. The bank has to take charge of end-of-record characters in this format to be provided for download.

All order types coded in ASCII (e.g. PKI, DKI, etc.) are defined with the ASCII end-of-line character X'0D0A' (which corresponds to the definition of SWIFT formats). The end-of-line characters have to be contained as ASCII code X'0D0A' prior to the encryption/compression of the order data.

#### **Order of procedures within VEU**

The bank system has to verify the correct order of the procedures within the VEU. Especially, HVU (or HVZ respectively) is a necessary precondition for the following steps.

#### **Length of lines of the customer protocol**

According to the requirements, the length of lines has to be limited to 72 characters.

#### **Handling of orders not completely authorized**

The ZKA's recommendation for the conditions of remote data transfer has stipulated in appendix 1a, chapter 3.1, that orders not completely authorized are to be deleted after the period of time that has been agreed upon with the customers.

**13 Utilised standards and references**

<b>Standard</b>	<b>Title</b>	<b>References</b>
PKCS#5 v2.0	Password-Based Encryption Standard.	<a href="ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2-0.pdf">ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2-0.pdf</a> , <a href="ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2-0.asn">ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2-0.asn</a>
PKCS#8 v1.2	The Private Key Information Syntax Standard.	<a href="ftp://ftp.rsasecurity.com/pub/pkcs/ascii/pkcs-8.asc">ftp://ftp.rsasecurity.com/pub/pkcs/ascii/pkcs-8.asc</a> , <a href="ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-8/pkcs-8v1_2.asn">ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-8/pkcs-8v1_2.asn</a>
PKCS#11 v2.20	Cryptographic Token Interface Standard	<a href="ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf">ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf</a> , <a href="ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20errata.txt">ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20errata.txt</a>
PKCS#12 v1.10	Personal Information Exchange Syntax Standard	<a href="ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf">ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf</a> , <a href="ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12-tc1.pdf">ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12-tc1.pdf</a>

## 14 Appendices

### 14.1 Appendix 1 – Example of the computation of a hash value

The file input.xml provides an example for the computation of a hash value for the authentication signature which is available here in printed form. Moreover, the file can be downloaded from the Internet: [www.ebics.de](http://www.ebics.de), „Specification“:

The example was created on the basis of XSD H001.

```
<?xml version="1.0" encoding="UTF-8"?>
<ebicsResponse xmlns="http://www.ebics.org/H001"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ebics.org/H001
http://www.ebics.org/H001/ebics_response.xsd"
  Version="H001" Revision="1">
  <header authenticate="true">
    <static>
      <TransactionID>F66F07FE33E167DDEBD165F159A05494</TransactionID>
      <NumSegments>1</NumSegments>
    </static>
    <mutable>
      <TransactionPhase>Initialisation</TransactionPhase>
      <SegmentNumber lastSegment="true">1</SegmentNumber>
      <ReturnCode>000000</ReturnCode>
      <ReportText>[EBICS_OK] OK</ReportText>
    </mutable>
  </header>
  <AuthSignature>
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <ds:Reference URI="#xpointer(//*[@authenticate='true'])">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>
zvDGDhcFikkAFh09A0xhCUI7q/s=
```

## EBICS specification

EBICS – Implementation Guide Version 1.7

---

```
</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>
YfyIWdz6BpYfD4cp+8AEG9vtpbERO0KZ5uP7IbF9CsxE3cMdcSrFukKWPPh7inI3MkgSgF5zo
3rWc3vYSHXwvGyN/J4397cNdEgB+qtZT26f8JU4MezEDwaNcLD8vLAe3Jv3S0Y5x4jI8NVdW
BolgpE107JuFgKKwFpxmlDc/ZBY=
</ds:SignatureValue>
</AuthSignature>
<body>
<DataTransfer>
<DataEncryptionInfo authenticate="true">
<EncryptionPubKeyDigest Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"
Version="E001">
36LLyy9+tfm2tc2xS3T/RBRDbcs=
</EncryptionPubKeyDigest>
<TransactionKey>
VfR/MnTvOQzo6z/KCXLEINcSciQdkukWAncFsqB2wVyu79Z7dqhAM2qTryZZADIXrGEJVZ/0
tw0Dn1hnaWRfUWznEzGVffh3wPBVx6h+85pnpoyeLDqppxD9AbeGG5n8RPbQBEdTXYuzrGS2
4HyhyYH3/+Eegl4U6RsLnIHJcy0=
</TransactionKey>
</DataEncryptionInfo>
<OrderData>
rUzGiGNHIOV0eTM6MaJPsoQjZrUIhQVPsKE4pXh30BP2sqNo/e5/WsBi+bVArj3GPciGk+Nc
LmaeVHLIiwe2RV18wEnWMqR3mcGsi+0ZIGAsdNqGlpJt5xmKoETC7OMvBIKq9TfIK0WepnnK
7eeDzVwGAmwBzgo4IIKQWEHkj8pK587R4zd4X4o25uFWQLNex6gOIUwBRkrTY0BF0+2KTD+C
JK/XMGF4zOlRrPryj/HvrwWhm+G7X5vfsPSa1ou+5sl71WCfexE7tbjG5T+eGTyKuWhUmutV
l0XbFfdj8LEJdTguPs/c5utBrZaaO4Mpc1ug7XWpvEFBZ2W+jNQe9E9BTT8rQAloojWQSzQU
ZtOTJkLzZlgM/2n+vjrUdzjKD4FvdMPvWuC7sWk1/qeD10NgaaTfQf/xKZHxPArHEYN/GP6T
iUl1mdqxLSuXNK105X3pRuoYEQ6kRu8r0iErtaSeqzS+VSOpfX0s4s+wulMWTAAQ53tW3UWw
U2Y02rqlFns61+8rdcwYChYYgZEvTJo1lkqbfZLTNqd147dHbpSONRbz8pQgNorilyey+z6M
24dHRxoxtdTYwC+L/k6tZNaiQih4s9TunL2jQ8VO8YfXdtBCvt9a4XAkzZy4XhAg1gdZq/V
jpZvZ4E2csY2nPP7OrCTYgxuYscMia7ZzOi/xGX/wAnR/xLBrmQoJaDj3JGmfACgGe2/xM4+
Hlmy+NB+M3Md8fYKOYNADdQ6VGI4wfsibhJEHcfmWDsM8RbTYuJQjSdip0fZbGVV9op6Pvmx
+U3qQki+E/PxwFPHyyONUftUSy1cbdDo6NP+VgWQBfKRodAEtMLOW4x7fBHDjMtlInIPcyHKd
N+mEQJsMniZe+Fi3V+8Q/PnrN3/2uks2k+buMCMX+BtZ8seoM7xzMIJ4+EHH0+Y3EK4SPfnY
HuGUDDeuEh9gaNq2i081dam2efSVTOdL6qisTd/MMetwTYNJ4JvRIBpo
</OrderData>
</DataTransfer>
```

## EBICS specification

EBICS – Implementation Guide Version 1.7

---

```
<ReturnCode authenticate="true">000000</ReturnCode>
</body>
</ebicsResponse>
```

Afterwards, the hash value is computed by means of the file hash\_output.bin and

```
<header xmlns="http://www.ebics.org/H001" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" authenticate="true">
  <static>
    <TransactionID>F66F07FE33E167DDEBD165F159A05494</TransactionID>
    <NumSegments>1</NumSegments>
  </static>
  <mutable>
    <TransactionPhase>Initialisation</TransactionPhase>
    <SegmentNumber lastSegment="true">1</SegmentNumber>
    <ReturnCode>000000</ReturnCode>
    <ReportText>[EBICS_OK] OK</ReportText>
  </mutable>
</header><DataEncryptionInfo xmlns="http://www.ebics.org/H001"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" authenticate="true">
  <EncryptionPubKeyDigest Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"
Version="E001">
36LLyy9+tfm2tc2xS3T/RBRDbcs=
  </EncryptionPubKeyDigest>
  <TransactionKey>
VfR/MnTvOQzo6z/KCXLEINcSciQdkukWAncFsqB2wVyu79Z7dqhAM2qTryZZADIXrGEJVZ/0
tw0Dn1hnaWRfUWznEzGVffh3wPBVx6h+85pnpoyeLDqppxD9AbeGG5n8RPbQBEdTxyuzrGS2
4HyhyYH3/+Eegl4U6RsLnIHJcy0=
  </TransactionKey>
</DataEncryptionInfo><ReturnCode xmlns="http://www.ebics.org/H001"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" authenticate="true">000000</ReturnCode>
```

returns the result:

```
CEF0C60E17058A4900161D3D034C6109423BABFB
```

Start and end of file are marked by angle brackets. All line feeds are independent of the input file and consist of line feed (without carriage return) only. Re-formatting within tags do not change the hash value. Empty elements

## EBICS specification

EBICS – Implementation Guide Version 1.7

---

<mutable/>

are always converted into opening and closing tags

<mutable></mutable>

Therefore, the file hash\_input2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ebicsResponse xmlns="http://www.ebics.org/H001"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ebics.org/H001
http://www.ebics.org/H001/ebics_response.xsd"
  Version="H001" Revision="1">
  <header authenticate="true">
    <static>
      <TransactionID>F66F07FE33E167DDEBD165F159A05494</TransactionID>
      <NumSegments>1</NumSegments>
    </static>
    <mutable>
      <TransactionPhase>Initialisation</TransactionPhase>
      <SegmentNumber lastSegment="true">1</SegmentNumber>
      <ReturnCode>000000</ReturnCode>
      <ReportText>[EBICS_OK] OK</ReportText>
    </mutable>
  </header>
  <!-- Ignore me -->
  <AuthSignature><ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#xpointer(//*[@authenticate='true'])">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      </ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>
zvDGDhcFikkAFh09A0xhCUI7q/s=
      </ds:DigestValue>
    </ds:Reference>
```

## EBICS specification

EBICS – Implementation Guide Version 1.7

---

```
</ds:SignedInfo>
<ds:SignatureValue>
YfyIWdz6BpYfD4cp+8AEG9vtpbERO0KZ5uP7IbF9CsxE3cMdcsrFukKWPh7inI3MkgSgF5zo
3rWc3vYSHXwvGyN/J4397cNdEgB+qtZT26f8JU4MezEDwaNcLD8vLAe3Jv3S0Y5x4jl8NVdW
BolgpE107JuFgKKwFpxmlDc/ZBY=
</ds:SignatureValue></AuthSignature>
<body>
<DataTransfer>
<DataEncryptionInfo authenticate="true">
<EncryptionPubKeyDigest xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns="http://www.ebics.org/H001"
Version="E001"
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"
>
36LLyy9+tfm2tc2xS3T/RBRDbsc=
</EncryptionPubKeyDigest>
<TransactionKey>
VfR/MnTvOQzo6z/KCXLEINcScIQdkukWAncFsqB2wVyu79Z7dqhAM2qTryZZADIXrGEJVZ/0
tw0Dn1hnaWRfUWznEzGVffh3wPBVx6h+85pnpoyeLDqppxD9AbeGG5n8RPbQBEdTXYuzrGS2
4HyhyYH3/+Eegl4U6RsLnIHJcy0=
</TransactionKey>
</DataEncryptionInfo>
<OrderData>
rUzGiGNHIOV0eTM6MaJPsOQjZrUIhQVPsKE4pXh30BP2sqNo/e5/WsBi+bVArj3GPciGk+Nc
LmaeVHLliwe2RV18wEnWMqR3mcGsi+0ZIGAsdNqGlpJt5xmKoETC7OMvBIKq9TfIK0WepnnK
7eeDzVwGAmwBzgo4IIKQWEHkj8pK587R4zd4X4o25uFWQLNex6gOIUwBRkrTY0BF0+2KTD+C
JK/XMGF4zOlRrPryj/HvrwWhm+G7X5vfsPSa1ou+5sl71WCfexE7tbjG5T+eGTyKuWhUmutV
l0XbFfdj8LEJdTguPs/c5utBrZaaO4Mpc1ug7XWpvEFBZ2W+jNQe9E9BTT8rQAlloojWQSzQU
ZtOTJkLzzlgM/2n+vjrUdzjKD4FvdMPvWuC7sWk1/qeD10NgaaTfQf/xKZHxPArHEYn/GP6T
iUI1mdqxLSuXNK105X3pRuoYEQ6kRu8r0iErtaSeqzS+VSOpfX0s4s+wulMWTQAQ53tW3UWw
U2Y02rqlFns61+8rdcwYChYYgZEvTJo1lkqbfZLTNqd147dHbpSONRbz8pQgNorilyey+z6M
24dHRxoxtdTYwC+L/k6tZNaiQih4s9TunL2jQ8VO8YfXdtBCvt9a4XAkzZy4XhAg1gdZq/V
jpZvZ4E2csY2nPP7OrCTYgxuYscMia7ZzOi/xGX/wAnR/xLBrmQoJaDj3JGmfACgGe2/xM4+
Hlmy+NB+M3Md8fYKOYNADdQ6VGI4wfsibhJEHcfmWDsM8RbTYuJQjSdip0fZbGVV9op6Pvmx
+U3qQki+E/PxwFPHyyONUfUSy1cbdDo6NP+VgWQBfKRodAEtMLOW4x7fBHDjMtlInIPcyHKd
N+mEQJsMniZe+Fi3V+8Q/PnrN3/2uks2k+buMCMX+BtZ8seoM7xzMIJ4+EHH0+Y3EK4SPfnY
HuGUDDeuEh9gaNq2l081dam2efSVTOdL6qisTd/MMetwTYNJ4JvRIBpo
</OrderData>
</DataTransfer>
```

## EBICS specification

EBICS – Implementation Guide Version 1.7

---

```
<ReturnCode authenticate="true"
>000000</ReturnCode>
</body>
</ebicsResponse>
```

returns the identical hash value.

## 14.2 Appendix 2 – Further Examples

### 14.2.1 Clarification of the Term “Technical Subscriber“

The following examples are to clarify typical actions of the customer server involving a technical subscriber and one or more human subscribers:

PartnerID of the customer	CUSTOM1
UserID (field SystemID) of the technical subscriber	TECHS1
UserID of the user = human subscriber	HUMANS1, HUMANS2, HUMANS3

case 1: The (human) subscriber initiates the download of account statements (STA) via a technical subscriber		
<b>Contents of the fields in the EBICS request</b>		<b>Authorisation order type</b>
<b>PartnerID</b>	CUSTOM1	
<b>UserID</b>	HUMANS1	STA
<b>SystemID</b>	TECHS1	

case 2: The technical subscriber is due to download account statements (e.g. via order type STA) automatically, e.g. overnight		
<b>Contents of the fields in the EBICS request</b>		<b>Authorisation order type</b>
<b>PartnerID</b>	CUSTOM1	
<b>UserID</b>	TECHS1	STA
<b>SystemID</b>	TECHS1	

case 3: HVU overview is based on (human) users (e.g. HUMANS1) because only (human) subscribers may possess and provide bank-technical ES's which are essential to complete orders.		
<b>Content of the fields in the EBICS request</b>		<b>Authorisation order type</b>
<b>PartnerID</b>	CUSTOM1	
<b>UserID</b>	HUMANS1	HVU
<b>SystemID</b>	TECHT1	

## EBICS specification

EBICS – Implementation Guide Version 1.7

---

case 4:  
The (human) subscriber initiates the upload of a payment file (e.g. via order type IZV) and subscribes it by his own (if further signatures are necessary for processing of the order, it is stored intermediately in the Distributed Electronic Signature (VEU)).

Content of the fields in the EBICS request		Authorisation order type
PartnerID	CUSTOM1	
UserID	HUMANS1	IZV
SystemID	TECHS1	
ES geleistet von	HUMANS1 (T/E/A/B)	IZV

case 5:  
The (human) subscriber initiates the upload of a payment file (e.g. via order type IZV) which was subscribed by other (human) subscribers (if further signatures are necessary for processing of the order, it is stored intermediately in the Distributed Electronic Signature (VEU)).

Contents of the fields in the EBICS request		Authorisation order type
PartnerID	CUSTOM1	
UserID	HUMANS1	IZV
SystemID	TECHS1	
ES provided by	HUMANS2 (T/E/A/B)	IZV
	HUMANS3 (T/E/A/B)	IZV

case 6:  
The technical subscriber initiates the upload of a payment file (e.g. via order type IZV) which was subscribed by other (human) subscribers (if further signatures are necessary for processing the order, it is stored intermediately in the Distributed Electronic Signature (VEU)).

Contents of the fields in the EBICS request		Authorisation order type
PartnerID	CUSTOM1	
UserID	TECHS1	IZV
SystemID	TECHS1	
ES provided by	HUMANS2 (T/E/A/B)	IZV
	HUMANS3 (T/E/A/B)	IZV

case 7:  
The (human) subscriber HUMANS1 initiates the intermediate storage of an IZV file in the Distributed Electronic Signature (VEU) and subscribes it by his own.

Contents of the fields in the EBICS request		Authorisation order type
PartnerID	CUSTOM1	
UserID	HUMANS1	IZV
SystemID	TECHS1	

## EBICS specification

EBICS – Implementation Guide Version 1.7

---

<b>EU provided by</b>	HUMANS1 (T/A/B)	IZV
-----------------------	-----------------	-----

Fall 8:

The technical subscriber TECHS1 automatically initiates the intermediate storage of an IZV file in the Distributed Electronic Signature (VEU).

<b>Contents of the fields in the EBICS request</b>		<b>Authorisation order type</b>
<b>PartnerID</b>	CUSTOM1	
<b>UserID</b>	TECHS1	IZV
<b>SystemID</b>	TECHS1	
<b>EU provided by</b>	TECHS1 (T)	IZV

### 14.2.2 Example for the Interpretation of Field AccountInfo@ID in the Order Types HKD and HTD

To clarify the use of AccountInfo@ID an XML example was suitably shortened:

```
<PartnerInfo>
```

```
  <AccountInfo>
```

```
    <AccountNumber> 111 </AccountNumber>
```

```
  </AccountInfo>
```

```
  <AccountInfo>
```

```
    <AccountNumber id="accid_2"> 222 </AccountNumber>
```

```
  </AccountInfo>
```

```
  <AccountInfo>
```

```
    <AccountNumber id="accid_3"> 333 </AccountNumber>
```

```
  </AccountInfo>
```

```
</PartnerInfo>
```

```
<UserInfo>
```

```
  <UserID> USER_1 </UserID>
```

```
  <Permission>
```

```
    <OrderTypes>IZG</OrderTypes>
```

```
  </Permission>
```

```
  <Permission>
```

```
    <OrderTypes>IZL</OrderTypes>
```

```
    <AccountID>accid_3</AccountID>
```

```
  </Permission>
```

```
</UserInfo>
```

In the example mentioned above the user USER\_1 may use order type IZG in combination with all accounts (111, 222 and 333). Furthermore USER\_1 may use order type IZL only for account 333.