9 Standard Guide for Implementing HL7 Communication Security

Bernd BLOBEL, Volker SPIEGEL, Peter PHAROW, Kjeld ENGEL and Rolf KROHN
Otto-von-Guericke University, Medical Informatics Department, Magdeburg, Germany

(approved as an HL7 Informative Document in September 1999)

9.1 Scope

This Standard Guide gives the framework and implementation details for implementing secure end-to-end EDI communication focusing on HL7. It is based on the “Standard Guide for EDI (HL7) Communication Security” and addresses system implementers.

Starting with an introduction to security services and general requirements for applications and especially for communication security in chapter 9.6, the fundamental security services needed as strong mutual authentication are described as well as securing information exchange by securing control data and message data, mentioning possible security attacks, security requirements and proposed implementations of solutions. In that context, different exchange protocols are considered. Regarding accountability, different non-repudiations services are discussed in detail.

The principles are illustrated in detail for a Secure File Transfer Protocol (SFTP), implemented in the authors’ environment.

9.2 Conformance

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, “OPTIONAL” used in this document have to be interpreted as described in RFC2119.

9.3 Normative References


### 9.4 Notation and Abbreviation

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>DN</td>
<td>Distinguished Name</td>
</tr>
<tr>
<td>EDI</td>
<td>Electronic Data Interchange</td>
</tr>
<tr>
<td>ESS</td>
<td>Enhanced Security Services (part of S/MIME Version 3)</td>
</tr>
<tr>
<td>FIPS PUB</td>
<td>Federal Information Processing Standards Publication (by NIST)</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>HL7</td>
<td>Health Level Seven</td>
</tr>
<tr>
<td>IESG</td>
<td>Internet Engineering Steering Group</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunication Union</td>
</tr>
<tr>
<td>MIME</td>
<td>Multipurpose Internet Mail Extension</td>
</tr>
<tr>
<td>MOSS</td>
<td>MIME Object Security Services</td>
</tr>
<tr>
<td>NRO</td>
<td>Non-Repudiation of Origin</td>
</tr>
<tr>
<td>NRR</td>
<td>Non-Repudiation of Receipt</td>
</tr>
<tr>
<td>PGP</td>
<td>Pretty Good Privacy</td>
</tr>
<tr>
<td>PKCS</td>
<td>Public Key Cryptography Standard</td>
</tr>
<tr>
<td>RFC</td>
<td>Request For Comments</td>
</tr>
<tr>
<td>S/MIME</td>
<td>Secure/MIME</td>
</tr>
</tbody>
</table>

### 9.5 Introduction

Establishing secure communication of EDI messages requires the selection and implementation of appropriate security services like authentication (user, application and system), integrity, confidentiality, and accountability (in the sense of non-repudiation of origin and receipt), as described and defined in the “Standard Guide for EDI (HL7) Communication Security” [7]. Communication security is related to the cryptographic enhancement of the whole message, such as message signing and message encryption, regardless of its internal structure.

Following the recommendations of the “Standard Guide for EDI (HL7) Communication Security” the solution for secure communication of EDI messages presented in this standard guide is based upon public key cryptography within a proper security infrastructure (public key infrastructure, PKI).

In the first section, the fundamentals of the solution are described independently of the communication protocol used. For each security service selected, the specific structure and contents of tokens exchanged between client and server is described. This includes all security services proposed in the “Standard Guide for EDI (HL7) Communication Security” as strong mutual authentication, integrity and confidentiality assurances, as well as non-repudiation of origin and receipt.

Subsequently, the communication protocol for token exchange is presented in detail, serving as the most convenient mechanism next to the paradigm of EDI communication in client/server architectures. This protocol is called the secure file transfer protocol (SFTP).
and is a security enhanced version of the unsecured RFC0959 compliant FTP.

9.6 Security Services and General Realisation

After giving some basic information about the public key infrastructure and the notation used in this standard guide, this section selects the security services needed for the control and data connection regarding the “Standard Guide for EDI (HL7) Communication Security”. Then, for each service, the general realisation is given independently of the communication protocol used. Possible attacks and countermeasures are considered, and the resulting structure and contents of the tokens exchanged are presented.

9.6.1 Fundamentals and Notation

This approach is based upon public key cryptography using asymmetric security techniques and symmetric security techniques as well. The latter is only taken for bulk data encryption within hybrid encryption employing a symmetric session key. Trust is established by a public key infrastructure (PKI) using trusted public keys certified by a certification authority (CA). For this purpose, X.509 certificates that are stored and managed in X.500 directories are applied.

Different key pairs MUST be used for authentication, digital signature generation/verification and encryption/decryption to avoid security compromise by possibly adaptive chosen-text attacks where the intruder chooses challenges to extract information about the key. This MAY be possible since the private key is used to compute the response and, thus, may reveal partial information. Hence, there are cryptographic needs for key separation requiring the use of one key for each purpose. For key separation, the notations given in Table 33 are used in the following.

The main symbol indicates the type of asymmetric key (PrK for private key and PK for public key), whereas the upper index denotes the key usage (auth for authentication, digSig for digital signature generation/verification, and crypt for encryption/decryption) and the lower index identifies the owner of the key (client for client, server for server and ca for the CA).

<table>
<thead>
<tr>
<th>Notation</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrK\auth</td>
<td>Private key for authentication (generation of digital signature)</td>
</tr>
<tr>
<td>PrK\digSig</td>
<td>Private key for integrity service (generation of digital signature)</td>
</tr>
<tr>
<td>PrK\crypt</td>
<td>Private key for decryption</td>
</tr>
<tr>
<td>PK\auth</td>
<td>Public key for authentication (verification of digital signature)</td>
</tr>
<tr>
<td>PK\digSig</td>
<td>Public key for integrity service (verification of digital signature)</td>
</tr>
<tr>
<td>PK\crypt</td>
<td>Public key for encryption</td>
</tr>
</tbody>
</table>
Strong security measures MUST be achieved by using strong cryptographic mechanisms and public key certificates, following X.509, that MUST be verified before usage every time. For verification, the public key certificate of the CA for digital signature verification (PKCS\textsubscript{CA}) is needed. This certificate MUST be checked itself before usage. Certificate verification MAY involve directory or local cache access performed prior to the authentication exchange.

For token formatting, the tag-length-value (TLV) format MUST be applied. Each token field is preceded by a tag-byte specifying the type of data and a length-word (little-endian order: first is low byte) that determines the amount of data that follows, as shown in Table 34. The concatenation of tokens is expressed by "||".

<table>
<thead>
<tr>
<th>Token Offset</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>TAG-byte (identifies the type of data)</td>
</tr>
<tr>
<td>0x0001</td>
<td>LEN-byte (low-byte data amount)</td>
</tr>
<tr>
<td>0x0002</td>
<td>LEN-byte (high-byte data amount)</td>
</tr>
<tr>
<td>0x0003</td>
<td>DATA-bytes (data)</td>
</tr>
<tr>
<td>and following</td>
<td></td>
</tr>
</tbody>
</table>

As XML becomes increasingly important, the TLV encoding format MAY be easily replaced by defining new XML attributes. The TLV-TAG-byte is transformed to an XML-tag, the values follow immediately, and the length is given implicitly by the XML-ending-tag. All authentication and control connection messages are then XML-messages. In addition, XML-messages can be sent over the data connection, transforming e.g. S/MIME or PKCS#7 into XML. The client and the server must have an XML generator and parser as well.

9.6.2 Strong Mutual Authentication

HL7 realises event-driven exchange of messages between Healthcare applications. Depending on specific circumstances or protocols used, the communicating principals may also include the human user of an information system. As a basic requirement, the communicating principals MUST be authenticated mutually. Because the solution implemented is intended to be open and flexible and also allow inter-protocol communication, different use cases must be reflected. In that context, the solution has to be fitted into the security environment, e.g., the European security infrastructure.

Following the “Standard Guide for EDI (HL7) Communication Security,” asymmetric techniques are applied for strong authentication. Before mutual trust between client and server can be established (the client is authenticated to the server and vice versa), the human user must authenticate oneself to a cryptographic module of the local end system (user authentication), which then acts as an initiator (client) and performs the digital signature generation and verification on behalf of the human user towards the responder (server), carrying out system authentication.

As recommended in the “Standard Guide for EDI (HL7) Communication Security” in chapter 8, human user authentication SHOULD be carried out by ISO/IEC7816 compliant
chip cards (e.g. Health Professional Cards = HPC) in combination with a PIN code. During a communication session, the chip card needs to be kept inserted in the chip card terminal for timed chip card request. When removing the chip card, the application MUST inhibit further operations and only continue to work if the chip card is inserted again and the subsequent user authentication is performed successfully.

The strong authentication of an initiator to the responder depends on the successful verification of the initiator’s digital signature binding with its key pair and also on a successful verification of the initiator’s digital signature (signing means showing the possession of the secret key) on a random number challenge generated by the responder. Accordingly, for mutual authentication the successful authentication of the responder to an initiator is checked. The binding of a principal’s unique identifier (i.e. the distinguished name (DN)) with its key pair is essential for proving the authenticity of its identity and must be checked prior to any authentication exchange. This is achieved by user authentication following verification of X.509 public key certificates retrieved from a X.500 directory.

Protocols for strong mutual authentication using asymmetrical security techniques can be found in [4, 10, 31]. There are some differences between these three sources concerning the authentication data structure, particularly regarding what token fields (such as digital signature, random numbers, time stamps and DNs) are recommended or optional in each step and what fields are covered by the digital signature. The order of authentication (first, the client is authenticated to the server, and afterwards the server is authenticated to the client in turn) remains the same, of course. Mutual authentication is performed by a three-way challenge-response-protocol for security and efficiency reasons, limiting the amount of tokens exchanged.

The authentication procedures defined in [31] are intended to be used between directory user agents (DUAs), but mainly follow the other specifications. Random numbers are used in combination with time stamps

[10] serves as a basis for the authentication protocols defined in [4] and specifies different protocols addressing both unilateral and mutual entity authentication, which make usage of public key cryptography algorithms.

In [4] only one protocol for each unilateral and mutual authentication has been selected from [10] and certain authentication token fields and protocol steps are described in greater detail than in the ISO specification. Furthermore, [4] is less strict, allowing the arbitrary ordering of token fields. The appendices A through D of this specification contain several optional methods and sets of rules for formatting and encoding authentication information (ASN.1 Notation and CER/DER encoding, Simple Public Key GSS-API Mechanism (SPKM), formatting based on ANSI X9.26-1990 and Base64 encoding) helping to promote the interoperability of various implementations of the authentication protocols defined. These appendices are provided for informational purposes only and are not part of the standard. Formatting and encoding is left to the discretion of the implementers. Moreover, to avoid the use of synchronised clocks to verify the timeliness of authentication tokens, authentication exchanges using time stamps were not chosen for [4]. Beyond that, sequence numbers have not been chosen, due to their requirement of maintaining sequence number log tables. Instead, random number challenges are used for both time stamps and sequence numbers. Finally, biometric authentication techniques are not included, but discussed in several other papers.

For the protocol presented in the following pages, all standards and recommendations mentioned above are combined and enhanced, gaining as much robustness and security as possible.
9.6.2.1 Possible Attacks and Countermeasures

For carrying out strong mutual authentication using asymmetric techniques, authentication tokens have to be exchanged by a challenge-response protocol. In general, these tokens MUST be completely integrity-protected to detect alteration of any kind. Furthermore, data origin authentication as well as non-repudiation of origin and receipt MUST be offered by the authentication protocol.

To cover all the security services mentioned and to detect or prevent attacks on the protocol, the tokens exchanged MUST contain certain fields like random numbers, time stamps, DNs and others. In the following section, possible attacks are listed giving appropriate countermeasures resulting in mandatory token fields. Forms of attack include impersonation/masquerading, token manipulation, replay, relay/forced delay, interleaving, reflection, key-related and implementation-related.

**Impersonation** or **masquerading** is the representation or implication of a false identity (i.e. assuming the identity of one of the legitimate parties in the network). This threat is eliminated by the authentication service itself assuring that the identity claimed is authentic. This is achieved by binding the identity (DN) to the token sent using the integrity service to provide data origin authentication. Authenticity of a public key is given by a CA-created certificate that binds identity and public key together. Proving this authenticity means verifying the certificate and possibly a chain of certificates to establish a hierarchy of trust. For data origin authentication and non-repudiation of origin, the distinguished name (DN) and physical location of the sender MUST be included. The physical location is specified by IP address and network adapter hardware address (MAC).

**Token manipulation** is addressed by integrity protection of the complete token for all exchanges performed. **Continuity of authentication** MUST be assured by binding the authentication service and the integrity service for further token exchanges (see control data) so that no intruder can cut in or take over after the authentication has been completed. Moreover, system authentication MUST be performed at the beginning and throughout each session (timed authentication). For integrity protection, digital signatures MUST be applied. For each signed message, the signature MUST be included for verification purposes.

**Replay attacks** MUST be addressed by including pre-assigned random numbers generated by the counterpart that are checked for equality when the counterpart receives the reply. Moreover, a chaining of random numbers MUST be carried out to verify if the number received by the counterpart in the last message is the same as that which has come with the current message. As the same random number challenge MUST never be issued twice or accepted twice by the same machine (client, server), a random number log table MUST be maintained. Furthermore, a sequence number MUST be applied for each token. This number must be recorded as well to prevent duplications. To reduce logging of the unique numbers (random number, sequence number) to a certain time, window time stamps SHOULD be used as continuous transaction numbers. The time stamps of token generation and expiration time are included in the token applying UTC time (Universal Time Co-ordinated). Thus, a secure time service that offers synchronous clocks is needed. Without a secure time server, the time difference of the stamps can only be verified using local time. Different report logs have to be maintained by the server for each client.

A token identifier MUST be included to recognise and distinguish the different kinds of tokens exchanged for authentication (request, data1, data2, and data3).

To eliminate **relay attacks** where the intruder acts as a wire (i.e. forced delay and intruder-in-the-middle), additional measures MUST be applied. First, the continuity of
authentication is assured as described above. Furthermore, the DN, IP address and the MAC (determining the physical location) of the recipient are included (dedicated challenge). Then, the time stamps included prohibit delays that are longer than the time window given. Short time outs are applied for the communication protocol model (i.e. the temporal distance of commands and replies as well as of each command in relation to the next in a chain of commands is checked using short time windows). Lastly, the authentication tokens indicate the role of the issuer (initiator, responder). A state indicator (authentication invitation, authentication request) marks whether authentication is being invited by the verifier or requested by the initiator. This is needed if both parties can start the authentication procedure, which is generally the case for mutual authentication protocols. This additional information may not be included if the entity that initiates interaction is either always the claimant or always the verifier.

**Attacks on the implementation as interleaving** or **reflection** MUST be addressed also. Uni-directional keys, where each principal has a unique signing key must be used. The identifiers of the target party are included, and tag-length-value encoding (TLV, see Table 34) is applied for field identification. TLV encoding permits randomisation of the order of fields inside tokens and prevents message symmetries. Furthermore, TLV protects against an inconsistent view of token fields giving a unique standardisation of all possible contents. The fields are distinguishable from each other independent of their token position. Thus, the implementation is more efficient due to the unambiguousness gained.

Token IDs and sequence numbers protect against an inconsistent view of tokens giving each message a unique tag of position within a stream of messages.

Confidentiality MUST NOT be applied, otherwise **key attacks** are possible. Since the control data is a well-known and often a small set of commands resulting in short tokens, key attacks like forward search or dictionary may be successful. Moreover, applying unnecessary security services may result in security flaws due to possible interaction between security mechanisms. When used in isolation, security mechanisms provide an acceptable level of security, but may become more vulnerable when used in combination with other mechanisms (see [11], page 15).

NRR SHOULD be provided by including the hash value of the previously received token in the new message to be sent.

After the authentication (user and system) has been successfully performed, authorisation and audit based upon the user’s identity MAY be carried out. The identity involved is obtained from the authentication tokens (as the field containing the DN).

### 9.6.2.2 Proposed Implementation

In addition to the fields needed in the authentication tokens as given in the “Standard Guide for EDI (HL7) Communication Security” and explained above, all tokens MUST be

---

6 The selective combination of information from one or more previous or simultaneously ongoing protocol sessions, including possible origination of one or more protocol sessions by an intruder itself.

7 An interleaving attack involving sending information from an ongoing protocol session back to the originator of the information.

8 Applying forward search means that the intruder takes all $2^n$ possible entries of a token field and encrypts them using the public key of the original recipient and compares each of the $2^n$ cipher texts with the value in the transaction actually encrypted ($n$ denotes the amount of bits per token).

9 When performing dictionary attacks, the intruder encrypts all entries of a dictionary (e.g. the list of communication protocol commands) with the public key of the original recipient and compares the result to the transmitted value.
Base64-encoded and canonicalised afterwards before transmission for **system interoperability preventing loss of data bits** in environments not capable of full binary transport. Not applying these conversions may result in invalidation of the digital signature.

The resulting protocol scheme is shown in Figure 26 and the tokens exchanged (Authentication Request, AuthData1, AuthData2 and AuthData3) are presented within the sequence of the protocol. Each token field is **TLV**-formatted which is not shown explicitly.

It may be difficult for the client to obtain the MAC address or the DN of the destination server in order to build the authentication request. A possible solution is to store them in the client environment statically, but handling this becomes difficult when changing the Ethernet card or the name of the server. In such cases, these values **MAY** be omitted from the authentication request token (and the server does not check them). In this case, the client obtains these parameters in AuthData1 for further usage. However, the client **SHOULD** know the IP address of the server in order to establish a direct network connection. This address **MUST** be included in the request and checked by the server.

Among other things, the TLV enables unique identification of the values and free ordering. However, if a signature is calculated over some TLV-encoded items, the ordering **MUST** be the same for the verification process. Otherwise, the digital signature becomes invalid. To ensure equality of the encoded items on both the client and the server, the signature **MUST** be included either before or after all other data items signalling that, for verification purposes, the signature has to be calculated over all data following or all data before respectively.

![Figure 26: Strong Mutual Three-Way Authentication](image)

1. At first, the client initiates system authentication by sending an authentication request token to the server:
   a. Generate:
      \[
      \text{AuthReq}' = \text{SeqNo} \ || \ \text{TokenID} \ || \ \text{IP}_{\text{client}} \ || \ \text{MAC}_{\text{client}} \ || \ \text{DN}_{\text{client}} \ || \ \text{IP}_{\text{server}} \ || \ \text{MAC}_{\text{server}} \ || \ \text{DN}_{\text{server}} \ || \ \text{TS}_{\text{gen}} \ || \ \text{TS}_{\text{exp}} \ || \ \text{Role-Initiator} \ || \ \text{State-Request} \ || \ R_{\text{client}} \ || \ \text{Auth-Mechanism}.
      \]
   b. Calculate the digital signature over all fields:
      \[
      \text{DS}_1 = PrK_{\text{client}}^{\text{auth}} (\text{AuthReq}').
      \]
   c. Concatenate the digital signature and the generated token:
      \[
      \text{AuthReq}'' = \text{AuthReq}' \ || \ \text{DS}_1.
      \]
d. Perform Base64-encoding and canonicalisation afterwards:
   \[ \text{AuthReq} = (\text{AuthReq}')_{\text{Base64, Canon}}. \]
e. Send the token to the server.

2. On receipt of AuthReq, the server performs the following operations:
   a. Apply Base64-decoding.
   b. Check if all necessary fields are included (using the TAG-byte).
   c. Verify the certificate of \( \text{PK}_{\text{CA}} \), use \( \text{PK}_{\text{CA}}^{\text{sign}} \) to verify \( \text{PK}_{\text{client}}^{\text{auth}} \) and take \( \text{PK}_{\text{client}}^{\text{auth}} \) to check the digital signature of the token received.
   d. Check if the TLV-format is correct, i.e. if the values of length are correct matching the length of data supplied.
   e. Check token type (TokenID1), sequence number (SeqNo1), time stamps (\( \text{TS}_{\text{gen1}}, \text{TS}_{\text{exp1}} \)) as well as role and state indicator (Role-Initiator, State-Request) for validity.
   f. Verify the identifiers of sender and recipient (DNs, IP addresses and MACs).
   g. Evaluate the authentication request command (AUTH-Command).

3. Then, the server builds AuthData1:
   a. Generate:
      \[ \text{AuthData1}' = \text{SeqNo2} || \text{TokenID2} || \text{IP}_{\text{client}} || \text{MAC}_{\text{client}} || \text{DN}_{\text{client}} || \text{IP}_{\text{server}} || \text{MAC}_{\text{server}} || \text{DN}_{\text{server}} || \text{TS}_{\text{gen2}} || \text{TS}_{\text{exp2}} || \text{Role-Responder} || \text{State-Request} || \text{R}_{\text{client}}' || \text{R}_{\text{server}} || \text{HashValue}_{\text{AuthReq}}'. \]
   b. Calculate the digital signature over all fields:
      \[ \text{DS}_2 = \text{PrK}_{\text{server}}^{\text{auth}} (\text{AuthData1}') \]
   c. Concatenate the digital signature and the generated token:
      \[ \text{AuthData1}'' = \text{AuthData1} || \text{DS}_2. \]
   d. Perform Base64-encoding and canonicalisation afterwards:
      \[ \text{AuthData1} = (\text{AuthData1}'')_{\text{Base64, Canon}}. \]
e. Send the token to the client.

4. On receipt of AuthData1, the client performs the following operations:
   a. Apply Base64-decoding.
   b. Check if all necessary fields are included (using the TAG-byte).
   c. Verify the certificate of \( \text{PK}_{\text{CA}}^{\text{sign}} \), use \( \text{PK}_{\text{CA}}^{\text{sign}} \) to verify \( \text{PK}_{\text{server}}^{\text{auth}} \) and take \( \text{PK}_{\text{server}}^{\text{auth}} \) to check the digital signature of the token received.
   d. Check if the TLV-format is correct, i.e. if the values of length are correct matching the length of data supplied.
   e. Check token type (TokenID2), sequence number (SeqNo2), time stamps (\( \text{TS}_{\text{gen2}}, \text{TS}_{\text{exp2}} \)) as well as role and state indicator (Role-Responder, State-Request) for validity.
   f. Verify the identifier of sender and recipient (DNs, IP addresses and MACs).
   g. Check if \( \text{R}_{\text{client}} = \text{R}_{\text{client}}' \).
h. For NRR, check HashValue\textsubscript{AuthReq}.

5. Now, the client builds AuthData2:
   a. Generate:
      \[
      \text{AuthData2}' = \text{SeqNo}_3 \ || \ TokenID_3 \ || \ \text{IP}_{\text{client}} \ || \ \text{MAC}_{\text{client}} \ || \ \text{DN}_{\text{client}} \ || \ \text{IP}_{\text{server}} \ || \ \text{MAC}_{\text{server}} \ || \ \text{DN}_{\text{server}} \ || \ \text{TS}_{\text{gen}}_3 \ || \ \text{TS}_{\text{exp}}_3 \ || \ \text{Role-Initiator} \ || \ \text{State-Request} \ || \ \text{R}_{\text{client}1}'' \ || \ \text{R}_{\text{server}1}' \ || \ \text{R}_{\text{client}2} \ || \ \text{HashValue}_{\text{AuthData1}}.
      \]
   b. Calculate the digital signature over all fields:
      \[
      \text{DS}_3 = \text{PrK}_{\text{auth}}_{\text{client}} (\text{AuthData2}').
      \]
   c. Concatenate the digital signature and the generated token:
      \[
      \text{AuthData2}'' = \text{AuthData2}' \ || \ \text{DS}_3.
      \]
   d. Perform Base64-encoding and canonicalisation afterwards:
      \[
      \text{AuthData2} = (\text{AuthData2}''\text{Base64, Canon}).
      \]
   e. Send the token to the server.

6. On receipt of AuthData2, the server performs the following operations:
   a. Apply Base64-decoding.
   b. Check if all necessary fields are included (using the TAG-byte).
   c. Verify the certificate of PK\textsubscript{CA}, use PK\textsubscript{CA}\textsuperscript{digSig} to verify PK\textsubscript{auth}\textsubscript{client} and take PK\textsubscript{auth}\textsubscript{client} to check the digital signature of the token received.
   d. Check if the TLV-format is correct, i.e. if the values of length correctly match the length of data supplied.
   e. Check token type (TokenID\textsubscript{3}), sequence number (SeqNo\textsubscript{3}), time stamps (TS\textsubscript{gen3}, TS\textsubscript{exp3}) as well as role and state indicator (Role-Initiator, State-Request) for validity.
   f. Verify the identifiers of sender and recipient (DNs, IP addresses and MACs).
   g. Check if \text{R}_{\text{client}1} = \text{R}_{\text{client}1}'' and \text{R}_{\text{server}1} = \text{R}_{\text{server}1}'.
   h. For NRR, check HashValue\textsubscript{AuthData1}.

After successfully processing step h., the client is authenticated to the server.

7. Then, the server builds AuthData3:
   a. Generate:
      \[
      \text{AuthData3}' = \text{SeqNo}_4 \ || \ TokenID_4 \ || \ \text{IP}_{\text{client}} \ || \ \text{MAC}_{\text{client}} \ || \ \text{DN}_{\text{client}} \ || \ \text{IP}_{\text{server}} \ || \ \text{MAC}_{\text{server}} \ || \ \text{DN}_{\text{server}} \ || \ \text{TS}_{\text{gen}}_4 \ || \ \text{TS}_{\text{exp}}_4 \ || \ \text{Role-Responder} \ || \ \text{State-Request} \ || \ \text{R}_{\text{client}2}'' \ || \ \text{R}_{\text{server}1}'' \ || \ \text{R}_{\text{server}2} \ || \ \text{HashValue}_{\text{AuthData2}}'.
      \]
   b. Calculate the digital signature over all fields:
      \[
      \text{DS}_4 = \text{PrK}_{\text{auth}}_{\text{server}} (\text{AuthData3}').
      \]
   c. Concatenate the digital signature and the generated token:
      \[
      \text{AuthData3}''' = \text{AuthData3}' \ || \ \text{DS}_4.
      \]
   d. Perform Base64-encoding and canonicalisation afterwards:
      \[
      \text{AuthData3} = (\text{AuthData3}''')\text{Base64, Canon}.
      \]
e. Send the token to the client.

8. On receipt of AuthData3, the client performs the following operations:
   a. Apply Base64-decoding.
   b. Check if all necessary fields are included (using the TAG-byte).
   c. Verify the certificate of $PK^{\text{CA}}_{CA}$, use $PK^{\text{digSig}}_{CA}$ to verify $PK^{\text{auth}}_{server}$ and take $PK^{\text{auth}}_{server}$ to check the digital signature of the token received.
   d. Check if the TLV-format is correct, i.e. if the values of length correctly match the length of data supplied.
   e. Check token type (TokenID), sequence number (SeqNo), time stamps (TS_{gen}, TS_{exp}) as well as role and state indicator (Role-Responder, State-Request) for validity.
   f. Verify the identifier of sender and recipient (DNs, IP addresses and MACs).
   g. Check if $R_{client2} = R_{client2}'$ and $R_{server1} = R_{server1}''$.
   h. For NRR, check HashValue_{AuthData2}'.

   After successfully processing step h., the server is authenticated to the client.

An overview of the authentication tokens exchanged and the verification carried out for the random numbers is shown in Figure 27.

![Figure 27: Overview of the Authentication Tokens Exchanged](image-url)
9.6.3 Securing the Control Data

When user and system authentication have been performed successfully, the control connection MUST be integrity protected as required by the “Standard Guide for EDI (HL7) Communication Security”. Furthermore, data origin authentication, non-repudiation of origin (NRO), and non-repudiation of receipt (NRR) SHOULD be provided.

9.6.3.1 Possible Attacks and Countermeasures

Integrity protection, using digital signatures, MUST be applied during the whole session over all fields contained in a message of control data (token). This allows detected token manipulation and assures the continuity of authentication binding the authentication service and the integrity service so that no intruder can cut in or take over after the authentication has been completed. For each signed message, the signature must be included for verification purposes.

Replay attacks MUST be addressed by including pre-signed random numbers generated by the counterpart that are checked for equality when the counterpart receives the reply. Moreover, a chaining of random numbers MUST be carried out to verify if the number received by the counterpart in the last message is the same as that which comes with the current message (see Figure 28). Furthermore, a sequence number MUST be applied for each token. To reduce logging of the unique numbers (random number, sequence number) to a certain time window, time stamps SHOULD be used as continuous transaction numbers. The time stamps of token generation and expiration time are included in the token applying UTC time.

A token identifier MUST be included to distinguish control data that is sent from the client to the server, which contain commands, from control data that is transmitted from the server to the client, which contain reply codes.

For data origin authentication and non-repudiation of origin, the sender MUST include his distinguished name (DN), IP address and network adapter hardware address (MAC).

To eliminate relay attacks where the intruder acts as a wire (i.e. forced delay and intruder-in-the-middle) additional measures MUST be applied. First, the continuity of authentication is assured as described above. Furthermore, the DN, IP address and the MAC (determining the physical location) of the recipient are included. Then, the time stamps included prohibit delays that are longer than the time window given.

At last, short time outs are applied for the communication protocol model (i.e. the temporal distance of commands to replies as well as of a command to the next in a chain of commands is checked using short time windows.)

Attacks on the implementation as interleaving or reflection MUST be addressed also. Uni-directional keys are used (each principal has a unique signing key), the identifiers of the target party are included, and tag-length-value encoding (TLV) is applied for field identification. Token IDs and sequence numbers protect against an inconsistent view of tokens giving each message a unique tag of position within a stream of messages.

Confidentiality MUST NOT be applied, otherwise key attacks are possible. Since the control data is a well-known and often a small set of commands resulting in short tokens, key attacks like forward search or dictionary may be successful. Moreover, applying unnecessary security services may result in security flaws due to possible interaction between security mechanisms.

When used in isolation, security mechanisms provide an acceptable level of security, but may become more vulnerable when used in combination with other mechanisms (see
NRR SHOULD be provided by including the hash value of the previously received token in the new message to be sent.

9.6.3.2 Proposed Implementation

In addition to the fields needed in the authentication tokens as given in the Standard Guide and explained above, all tokens MUST be Base64-encoded and then canonicalised before transmission for system interoperability preventing loss of data bits in environments not capable of full binary transport. Not applying these conversions may result in invalidation of the digital signature.

The resulting token contents for the control data connection are given in the following. First, the generation and verification of the tokens are described in detail. Then, a general overview of the token exchanged within the communication protocol is shown regarding the continuity of authentication by resuming the authentication protocol given below (see Figure 28). Each token field is TLV-formatted, which is not shown explicitly.

Besides others, the TLV enables unique identification of the values and free ordering. However, if a signature is calculated over some TLV-encoded items, the ordering MUST be the same for the verification process. Otherwise, the digital signature becomes invalid. To ensure equality of the encoded items on both the client and the server, the signature MUST be included either before or after all other data items, signalling that for verification purposes, the signature has to be calculated over all data following or all data before, respectively.

In general, the token generation process (of command or reply codes) for every control data token looks like the following:

a. Generate:

Token'\_n = SeqNo \|| TokenID\_m \|| IP\_client \|| MAC\_client \|| DN\_client \|| IP\_server \|| MAC\_server \|| DN\_server \|| TS\_gen \|| TS\_exp \|| [command \veq replyCode] \|| randomNumbers \|| HashValue_{Token'(n-1)}.

b. Calculate the digital signature over all fields:

DS\_n = [PrK^\text{dig}_\text{client} \(\text{Token'}\_n\)) \veq \PrK^\text{dig}_\text{server} \(\text{Token'}\_n\)].

c. Concatenate the digital signature and the generated token:

Token''\_n = Token'\_n \|| DS\_n.

d. Perform Base64-encoding and canonicalisation afterwards:

Token_n = (Token''\_n)\text{Base64, Canon}.

e. Send the token to the [server \veq client].

Basically, the following steps are performed on receipt of the token for verification:

a. Apply Base64-decoding.

b. Check if all necessary fields are included (using the TAG-byte).

c. Verify the certificate of PrK^\text{dig}_\text{CA}, use PK^\text{dig}_\text{client} to verify [PK^\text{dig}_\text{client} \veq PK^\text{dig}_\text{server}] and use [PK^\text{dig}_\text{client} \veq PK^\text{dig}_\text{server}] to check the digital signature DS\_n of the token.

d. Check if the TLV-format is correct (i.e. if the value of length correctly matches the length of data supplied.)

e. Check the token ID (TokenID), sequence number (SeqNo) and time stamps (TS\_gen).
Verify the identifiers of sender and recipient (DNs, IP addresses and MACs).

Check the random numbers for equality (see Figure 28).

For NRR, check HashValueToken(n-1).

Evaluate the [command ∨ reply code].

Client

... ...

\[
\text{SeqNo}_3 \| \text{TokenID}_3 \| \text{IP}_{\text{client}} \| \text{MAC}_{\text{client}} \| \text{DN}_{\text{client}} \| \text{IP}_{\text{server}} \| \text{MAC}_{\text{server}} \| \\
\text{DN}_{\text{server}} \| \text{TS}_{\text{gen}3} \| \text{TS}_{\text{exp}3} \| \text{Role-Initiator} \| \text{State-Request} \| \text{R}_{\text{client}1}'' \| \\
\text{R}_{\text{server}1}'' \| \text{R}_{\text{client}2} \| \text{HashValue}_{\text{AuthData}1} \| \text{DS}_3
\]

Authentication Data 2

\[
\text{R}_{\text{client}2} = \text{R}_{\text{client}2}'
\]

\[
\text{DN}_{\text{server}} \| \text{TS}_{\text{gen}4} \| \text{TS}_{\text{exp}4} \| \text{Role-Responder} \| \text{State-Request} \| \text{R}_{\text{client}2}' \| \\
\text{R}_{\text{server}1}'' \| \text{R}_{\text{server}2} \| \text{HashValue}_{\text{AuthData}2} \| \text{DS}_4
\]

Authentication Data 3

\[
\text{R}_{\text{server}1} = \text{R}_{\text{server}1}''
\]

\[
\text{SeqNo}_5 \| \text{TokenID}_5 \| \text{IP}_{\text{client}} \| \text{MAC}_{\text{client}} \| \text{DN}_{\text{client}} \| \text{IP}_{\text{server}} \| \text{MAC}_{\text{server}} \| \\
\text{DN}_{\text{server}} \| \text{TS}_{\text{gen}5} \| \text{TS}_{\text{exp}5} \| \text{command} \| \text{R}_{\text{server}2}' \| \text{R}_{\text{client}2}'' \| \text{R}_{\text{client}3} \| \\
\text{HashValue}_{\text{AuthData}3} \| \text{DS}_5
\]

Command

\[
\text{R}_{\text{client}3} = \text{R}_{\text{client}3}'
\]

\[
\text{SeqNo}_6 \| \text{TokenID}_6 \| \text{IP}_{\text{client}} \| \text{MAC}_{\text{client}} \| \text{DN}_{\text{client}} \| \text{IP}_{\text{server}} \| \text{MAC}_{\text{server}} \| \\
\text{DN}_{\text{server}} \| \text{TS}_{\text{gen}6} \| \text{TS}_{\text{exp}6} \| \text{replyCode} \| \text{R}_{\text{server}2}'' \| \text{R}_{\text{server}3} \| \text{R}_{\text{client}3}'' \| \\
\text{HashValue}_{\text{Token}5} \| \text{DS}_6
\]

Reply Code

\[
\text{R}_{\text{server}2} = \text{R}_{\text{server}2}'
\]

... ...

\[
\text{HashValue}'5 \| \text{DS}_6
\]

Figure 28: Control Data Tokens Exchanged Regarding Continuity of Authentication

9.6.4 Securing the Message Data

As described in the “Standard Guide for EDI (HL7) Communication Security,” the communication protocol implementation MUST provide integrity protection. Furthermore, confidentiality and non-repudiation services (of origin and receipt) SHOULD be offered. The protocol has to be usable in any desired environment (such as HL7) for secure delivery of data files containing different types of data such as HL7, X12, xDT\(^{10}\), XML messages or binary data; data type independence. The message data MUST be character converted and canonicalised to prevent loss of data bits. Furthermore, for correct handling and feature

---

\(^{10}\) As ADT, BDT, GDT, LDT and more. Used for standardised message transfer in Healthcare in Germany.
negotiation, a cryptographic syntax MUST be used for encapsulation. Thus, to satisfy all these requirements, MIME-object security MUST be applied.

Security Multi-parts for MIME [19], S/MIME version 2 [29], S/MIME version 3 [30], MOSS [20] or PGP/MIME [22] are appropriate for this purpose. Informational examples for applying Security Multi-parts for MIME and S/MIME version 2 are given in Annex B and Annex C, respectively. Being independent of the cryptographic protocol syntax, any other desired cryptographic syntax can be added when offering the featured needed.

Each cryptographic protocol SHOULD be used in three different operation modes (besides plain text) according to the local security policy: signed-only, encrypted-only or signed-and-encrypted. These modes MUST be realised by applying MIME-object nesting. For bulk encryption (content encryption) a strong symmetric session key (having at least 112 significant key bits) MUST be used and MUST be secured by strong asymmetric techniques (preferably by RSA with 1024 bits and above) for transport (hybrid encryption). The session key algorithm MUST be selectable and a new key SHOULD be calculated for each message data transport. Switching between the cryptographic protocols and their operation modes SHOULD be performed easily by the human user.

For transmission of large files, data compression or delivery of raw cryptographic objects MAY be applied. For Security Multi-parts for MIME and S/MIME these raw objects are PKCS#7-based as PKCS#7-objects [28] or CMS-objects [30]. PGP/MIME is based upon PGP-objects, whereas MOSS is not bound to a specific syntax.

Compression of EDI messages MUST be done before encryption, after applying the digital signature if needed ([12] chapter 5.4.1). In general, EDI messages compress well, since there is much repetitive data in most of the messages. Applying compression before encryption strengthens cryptographic security since repetitious strings are reduced due to their redundancy. The MIME standards [13] do not define any content encoding concerning compression, but allow the definition of additional content fields (see chapter 9 of RFC2045). As presented in [12], an additional content field “Content-Encoding;” (following RFC2068 chapter 3.5 and 14.12 for HTTP1.1) may be inserted to convey compression information. If gzip (see RFC1952) is used, this looks like "Content-Encoding: gzip".

Transport of raw cryptographic objects (as PKCS#7, CMS or PGP) can be applied to avoid the cryptographic syntax overhead of MIME security as Base64-encoding, MIME headers and trailers. Raw objects of this kind MUST NOT be used for transport of EDI messages, because neither canonicalisation nor Base64-encoding is performed. Without MIME headers, no content handling and feature negotiation can be performed. Furthermore, NRR can be only provided for CMS-objects in combination with the Enhanced Security Services (ESS, [30]). Otherwise, there is no NRR support available for these raw objects. For NRR related issues see section 9.6.4.3.

9.6.4.1 Encapsulating EDI-messages in MIME

Before delivery of EDI messages using MIME security, the message MUST be Base64-encoded to prevent loss or manipulation of certain EDI characters (as the HL7 segment terminator) leading to invalidation of the digital signature. Furthermore, the message MUST be inserted into a MIME body for delivery that must also be canonicalised. On receipt, the MIME body MUST be canonicalised for signature validation and the message has to be Base64-decoded afterwards. Informational examples for applying Security Multi-parts for MIME and S/MIME version 2 are given in Annex B and Annex C, respectively.
Chapter 9

As mentioned above, the implementation can be used in any desired environment for delivery of any type of data. Data type independence means that the receiving application must be able to recognise the type of data received. For that reason, if inserting an HL7 message into a MIME body, a content-type identifying HL7 messages MUST be used. Thus, the content-type application/x-EDI-HL7 SHOULD be applied. Additional parameters (for example syntax and version) MAY be stated in the content-type to specify encoding rules, for instance. When operating in an HL7 environment, data type independence MUST NOT be attended to, since the HL7 interface definitely knows that only HL7 message data is sent between applications. For that reason when inserting HL7 messages, the specialised content-type application/x-EDI-HL7 need not be used, but the content-type chosen MUST be able to carry the additional parameters as well. Another possible solution is to map the HL7 message (including the additional protocol parameters mentioned above) into a X12 message using the standardised mapping rules, and to insert the result into the content-type application/EDI-X12 defined in [18]. Other content-types could not be used as they do not feature the additional protocol parameters mentioned above. MIME encapsulation of X12 and EDIFACT objects is specified in [18] using the content-types application/EDI-X12 and application/EDIFACT.

For delivery of EDI messages, general requirement for interoperable EDI and security related issues are found in chapter 8 in this book.

9.6.4.2 Encapsulating Signed MIME Messages for Transport

When transporting signed data using multipart/signed by Internet (http, mail) or end-to-end in non-MIME environments, gateways are generally not aware of MIME security and treat this content-type as multipart/mixed or also apply conversions to the MIME structure and its contents according to the local format. Thus, either the original message cannot be reconstructed and the signature cannot be verified, or the signature verification fails.

To counter this problem, [29] and [30] propose two solutions. Either the content-type application/pkcs7-mime or the content-type application/mime SHOULD be used to pass signed data through the gateway, intact, for an S/MIME facility. The major difference between these two alternatives is that the first uses a PKCS#7 object and the latter encapsulates the whole multipart/signed entity.

The encapsulation using application/mime has been also specified by [1], but this Internet-Draft is expired and has been deleted without publication as a RFC.

A description for secure EDI documents exchange using http transport is given in [14].

9.6.4.3 Non-Repudiation

According to the “Standard Guide for EDI (HL7) Communication Security,” non-repudiation of origin (NRO) and receipt (NRR) SHOULD be provided for the transmission of message data (EDI messages and other data files).

Generally, NRO MUST be provided by inserting information about the sender (in the role of the signer) as its distinguished name or public key (certificate).

For PKCS#7 and CMS, the signedData object MUST be used to assure NRO. This can be achieved by including certificates or authenticated attributes. For all PKCS#7 objects mentioned, certificates are included using the specific field called
ExtendedCertificatesAndCertificates for a set of PKCS#6 extended certificates and X.509 certificates (chains of certificates) or using the field ExtendedCertificateOrCertificate for either a PKCS#6 extended certificate or an X.509 certificate. For CMS-objects, certificates are included using the field certificates containing a collection of PKCS#6-certificates (obsolete for CMS) or X.509 (attribute) certificates.

Authenticated attributes are inserted in the attribute authenticatedAttributes of the field signerInfo for PKCS#7-objects, whereas the attribute signedAttrs is used for CMS-objects. For MOSS, the field Originator-ID: can hold the DN (including email if desired) or the public key of the sender (originator).

For providing NRR, signed receipts MUST be used. In general, NRR can be realised by the MIME-syntax itself or the cryptographic objects embedded. The way of providing NRR by MIME-syntax is given by [12, 21, 27] and is described in section 9.6.4.3.1. Following this scenario, NRR can be provided independently of the objects embedded. When using S/MIME version 3, NRR MUST be provided by the CMS-objects embedded in combination with the Enhanced Security Services (ESS) as defined by [30]. This scenario is described in section 9.6.4.3.2. There is no other way to offer NRR yet. No NRR support is available on the PKCS#7-level.

Since the return of message content MAY be wasteful of network bandwidth and time, an appropriate strategy SHOULD be chosen. Thus, only the hash value of the last message received SHOULD be included and not the full message itself.

9.6.4.3.1 NRR for MIME-Object Security Protocols

When using MIME-object security protocol as Security Multi-parts for MIME, S/MIME version 2, MOSS or PGP/MIME the following specifications and formats for receipts and signed receipts MUST be applied for provision of NRR. For S/MIME version 3, NRR MUST be implemented as given in section 9.6.4.3.2.

The format of requesting and the format of receipts are defined in [27]. The format of signed receipts and their requests are specified in [12] chapter 5. In order to request a signed receipt, the sender places the following headers before the first content-type of the message. The header Disposition-notification-to: contains the return address (usually mail address), Disposition-notification-options: as well as its parameter disposition-notification-parameters= specifies how and what (as protocol and message digest algorithm) message disposition notifications should be generated.

Receipts are built using the content-type multipart/report as defined in [21] that encloses bodies for textual status description (first body; for instance content-type text/plain), for message disposition notification (second body; MDN, namely the content-type message/disposition-notification) as specified in [27] and a" reference" to the original message (third body). For human diagnosis, the textual status description (first body part of multipart/report) can be used to include a more detailed explanation of the error conditions reported by the disposition headers. Following [21, 27] for receipts, the original message (if encrypted, in its encrypted form) or part of it (for instance received headers) should be included as a third body part (optional body part) or omitted if message headers are not available. Full message inclusion is only recommended if the request level is absent, otherwise partial inclusion is recommended. In any case, the reference is achieved by the field Original-Message-ID: in addition to
other fields like Reporting-UA:, Original-Recipient:, Final-Recipient: and Disposition: of the second body part without any security protection (for example: possible forger of MDNs).

| Content-Type: multipart/related;<CR><LF>  
| type="application/x-edi-response";<CR><LF>  
| boundary="<boundary1>";<CR><LF>  
| <CR><LF>  
| --<boundary1><CR><LF>  
| Content-Type: application/x-EDI-HL7<CR><LF>  
| Content-Transfer-Encoding: base64<CR><LF>  
| <CR><LF>  
| <base64-encoded EDI reply message><CR><LF>  
| --<boundary1><CR><LF>  
| Content-Type: multipart/report;<CR><LF>  
| report-type="disposition-notification";<CR><LF>  
| boundary="<boundary2>";<CR><LF>  
| <CR><LF>  
| --<boundary2><CR><LF>  
| Content-Type: text/plain<CR><LF>  
| Content-Transfer-Encoding: 7bit<CR><LF>  
| <CR><LF>  
| <some text describing the status><CR><LF>  
| --<boundary2><CR><LF>  
| Content-Type: message/disposition-notification<CR><LF>  
| Content-Transfer-Encoding: 7bit<CR><LF>  
| <CR><LF>  
| Reporting-UA: <ua-name>; <ua-identifying-string><CR><LF>  
| Final-Recipient: <address-type>; <generic-address><CR><LF>  
| Original-Message-ID: <message-id><CR><LF>  
| Disposition: <action-mode>/<sending-mode>;<CR><LF>  
| <disposition-type>/<disposition-modifier ><CR><LF>  
| Received-Content-MIC: <mic>, <micalg><CR><LF>  
| --<boundary2>--<CR><LF>  
| <CR><LF>  
| --<boundary1>--<CR><LF>  

Figure 29: Prototype of the multipart/related Content-type

Signed receipts are built following [12] using the content-type multipart/report as described above, but the Base64-encoded MIC (message integrity check or message digest) of the original plain text message is inserted into the new field Received-content-MIC: in the second body to establish the reference. For any signed messages (this means that signed/encrypted must be decrypted first), the MIC to be returned is
calculated on the canonicalised (multi-part) MIME header and content. For encrypted-only messages, the MIC to be returned is calculated on the decrypted and, afterwards, canonicalised (multi-part) MIME header and content.

For plain text messages the MIC must be calculated over the message contents before their transfer encoding and without any MIME or other headers. Returning the original or parts of the received message in the third body of multipart/report is not required (optional body part), but placing the received headers into that body is recommended. At last, the complete content-type multipart/report is signed after its canonicalisation using application/pkcs7-mime with smime-type=signed-data or multipart/signed for S/MIME version 2, or multipart/signed for secure MIME.

For validation, the MIC contained in multipart/report received from the server must be compared with the MIC calculated by the client.

For bundling purposes, the server's response, comprised of the reply message and the signed receipt (the whole content-type multipart/report as described above but unsigned and un-encrypted), are bound together by the content-type multipart/related [23]: The server computes a reply message and inserts this message into the MIME entity (for HL7 the content-type application/x-EDI-HL7). This entity is inserted as the first part of the multipart/related MIME entity. The multipart/report entity is inserted unsigned and unencrypted as the second body part. A prototype of the multipart/report entity is shown in Figure 29.

Then, the multipart/related entity (parameter type application/x-EDI-RESPONSE and consisting of two bodies) is canonicalised and then signed. If confidentiality is needed, the result itself can be enveloped.

To summarise, there are only two transactions between client and server (if the client abandons sending a MDN receipt for the server's response in turn): The client sends an request message including a request for a signed receipt and the server responds by transmitting the reply message and the receipt signed and encrypted as explained above.

9.6.4.3.2 NRR for S/MIME Version 3

When using the S/MIME version 3 as defined by [30], the Enhanced Security Services for S/MIME (ESS, [30]) MUST be used for providing NRR by signed receipts. The ESS use the CMS (Cryptographic Message Syntax) as defined by [30]. The CMS is derived from PKCS#7 version 1.5. Signed receipts may be requested only if a message is signed and can optionally be encrypted by the sender of the receipt.

As described in chapter 2 of the ESS specification, the request is indicated by adding the attribute receiptRequest to the authenticatedAttributes field of the Signer Info object for which the receipt is requested. The attribute receiptRequest consists of the fields signedContentIdentifier, receiptsFrom and receiptTo. The field signedContentIdentifier is used to associate the signed receipt with the message requesting the signed receipt by a unique message identifier. Entities which has been requested to return a signed receipt are noted in the field receiptsFrom. For each entity to whom the recipient should send the signed receipt, the message originator must provide the GeneralNames (usually the originator's name only) in the field recipientTo.
A signed receipt is a signedData object encapsulating the receipt object identifier and the attribute receipt (in 'encapContentInfo') that consists of the fields version (set to 1 for now), contentType, signedContentIdentifier and originatorSignatureValue. The object identifier from the contentType attribute of the original message is copied into the contentType field of the receipt attribute, and the value of the signedContentIdentifier is copied also. The signature digest (including the receiptRequest attribute) of the original signedData object is copied into the field originatorSignatureValue.

The field authenticatedAttributes of signerInfo (a field of signedData) contains the attributes messageDigest, msgSigDigest, contentType and other attributes (for example the signingTime) indicating the time the receipt was signed.

The receipt is signed and the digest is included in messageDigest, the digest value, calculated to verify the signature of the original signedData object, is included in msgSigDigest and the receipt object identifier is inserted into contentType. At last, all authenticated attributes are signed and the signature is included in signature of signerInfo.

The signedData object is then put into an application/pkcs7-mime body with the parameter type signed-receipt. If this object should be encrypted within an envelopedData object, then an outer signedData object must be created encapsulating the envelopedData object, containing a contentHints attribute with the receipt object identifier as contentType. This is needed for the receiving agent to indicate that a signed receipt is contained within an envelopedData object.

To validate a signed receipt, the requestor must retain either the original signedData object, or the signature digest value of the original signedData object (contained in signature of signerInfo) and the digest value of the attribute receipt.

First of all, contentType, signedContentIdentifier and originatorSignatureValue are extracted from the receipt attribute to identify the original signedData object that requested the receipt.

Now, the digest of the original signedData object is compared with the value of msgSigDigest. If the originator has not retained the digest, it must be recalculated. If these values are identical, it is proven that the digest calculated by the recipient is based upon the received original signedData object including the same authenticatedAttributes containing the receiptRequest.

Then, the digest calculated by the originator for the receipt attribute is compared with the value of messageDigest. If the originator has not retained the digest, it must be recalculated. If these values are identical, it is proven that the recipient received the original signedData object signed by the originator to build the receipt attribute.

At last, the originator verifies the signature of the received signedData object (signature field of signerInfo) using the calculated digest of authenticatedAttributes. If the signature verification is successful, the integrity of the received signedData object containing the receipt attribute is proven and the identity of the recipient included in signerInfo is authenticated.
9.7 The Secure File Transfer Protocol (SFTP)

In this section, the communication protocol over TCP/IP-based networks is described that offers user and system authentication as well as a secure control and data connection according to the “Standard Guide for EDI (HL7) Communication Security.”

This is achieved by exchanging the tokens given in this guide (see section 9.6). This protocol is a security-enhanced version of the fundamental file transfer protocol given in [15] and is based solely on standards (e.g. ISO, NIST FIPS-PUB, ANSI and IETF/IESG RFCs). The protocol is called the secure file transfer protocol (SFTP).

File transfer of HL7 messages (batch processing) is carried out by transmitting one or more messages grouped in a file and encoded according to the encoding rules of HL7. Responses are grouped and transported similarly. According to communication security, SFTP wraps HL7 messages applying various selectable, cryptographic message syntax, such as PKCS#7, security multi-parts for MIME, S/MIME (version 2 or 3), MOSS or PGP/MIME. Security based on MIME takes advantage of the object-based features of MIME and allows secure messages. In general, SFTP is independent of the cryptographic syntax used; thus, any other syntax can be implemented without much effort.

Moreover, SFTP is able to process any desired type of file data as EDI messages, including EDIFACT, HL7, X12, xDT and others, or arbitrary binary data. Different operation modes (i.e. plain text, signed-only, encrypted-only or signed-and-encrypted) can be selected for message transmission according to the security policy given. Character encoding using the Base64-encoding scheme is selected and canonicalisation is applied for system interoperability, preventing loss of data bits that may lead to invalidation of the digital signature.

For establishing a public key infrastructure (PKI) using trusted public keys, all public keys are embedded into a certificate whose structure follows X.509, and the distinguished names (DN) used therein conform with X.501. The certificates are stored and managed in X.500 or LDAP directories.

9.7.1 The Protocol Model

The Secure File Transfer Protocol (SFTP) is based upon the TCP/IP protocol suite using the FTP client/server model, as defined in [15], regarding the additional requirements of [16] (chapter 4) that FTP implementations should follow. The TCP/IP protocol suite, compared to the OSI model, is presented in Figure 30.

An overview of the SFTP process model is shown in Figure 31. It is derived from the fundamental FTP model given in [15]. The protocol interpreter (PI) and the data transfer process (DTP) involved realise FTP processing by analysing and evaluating commands and replies (the part of the PI) as well as performing data transfer if needed (the part of the DTP). Thus, the PI is managing the control connection and the DTP is responsible for the data connection.
Basically, the SFTP process works like this. The server is listening on the well-known FTP service port (TCP port 21) waiting for a client connecting to that port. If the client performs a connection (from a dynamic port X), a so-called control connection is initiated that remains active for the whole session. On this connection, the client sends commands to the server and the server responds by sending reply codes using this connection in full-duplex operation mode. Normally, the control connection is closed by the client by sending an appropriate command (QUIT), but the server could also close the control connection in case of serious errors.

The data transfer is performed by establishing a second temporary connection in simplex operation mode. There are two modes for the establishment of such data connection:

1. In active mode, the client listens on a dynamic TCP port Y and sends a PORT command containing his IP address and port Y to the server, which then attempts to connect to that IP address and TCP port.
2. When using passive mode, the client sends a PASV command to the server, which listens on port 20 (or alternatively on a dynamic port) and informs the client where to connect by sending an appropriate reply code containing its IP address and TCP port.

As stated in [17], the passive mode should be preferred for firewall-friendly FTP. Switching between the active and passive data connection mode must be possible at any time.
All transfers (control and data connection) performed by the original RFC0959-FTP protocol are unsecured and have no security services such as strong authentication, confidentiality, integrity or accountability. Only simple authentication is carried out by transmitting the password in plain text using the USER and PASS command.

Looking at the process model described above, the enhancement of security for the FTP protocol MUST be located at the PI securing the control connection and at the DTP securing the data connection. Furthermore, before the client could perform any command (except the command to request authentication) and data transfer on the server, a strong mutual authentication MUST be performed between them. This is exactly the approach realised by SFTP. For the enhancement of security, many standard documents available are considered such as ISO Standards, IETF/IESG Internet Standards (RFCs), IETF Internet Drafts (IDs) and NIST publications (NIST FIPS PUB).

In addition, both client and server MUST apply timers to check if a connection is timed-out, that is, if the response or chained commands are out of time. This MUST be performed for the control and data connection as well as if the server is running on idle.

9.7.2 Strong Mutual Authentication

For user authentication, the human user SHOULD provide his or her HPC user name and PIN in combination with biometrics. After the SC has been opened successfully, all objects (e.g. keys) MUST be checked for completeness and validity (for instance certificates). During the SFTP session, the chip card needs to be kept inserted in the chip card terminal (timed chip card request). When removing the chip card, the application inhibits further operations and only continues to work if the chip card is inserted again and the user authentication that follows is successful.
Before the SFTP client can perform any command (except the command to request authentication) and data transfer on the server, strong mutual authentication MUST be performed between them as described in section 9.6.2.2.

As given in section 9.6.2.2, a unique identifier is included for each token exchanged to indicate its type and position in the exchange as shown in Figure 27. The following values SHOULD be used (in the style of [4] appendix A) in “byte” representation:

\[
\begin{align*}
\text{TokenID}_{\text{AuthReq}} &= 0x10 \\
\text{TokenID}_{\text{AuthData1}} &= 0x11 \\
\text{TokenID}_{\text{AuthData2}} &= 0x12 \\
\text{TokenID}_{\text{AuthData3}} &= 0x13
\end{align*}
\]

The sequence number SHOULD have the data type “word” (2 bytes, little endian order). Two time stamps SHOULD be included: one time stamp for token generation time and one for token expiration time. For time stamp generation, UTC time MUST be used and converted to seconds for the purpose of comparison (using “dword” (4 bytes, little endian order) representation). The time window MUST be of an appropriate length according to the physical properties of the underlying network (e.g. not smaller than 2 minutes). Role and state SHOULD have “byte” representation, all other items “string”.

For token formatting, the tag-length-value (TLV) format MUST be applied. The values used for the tag-byte of the TLV format (see Table 34) are presented in Table 35 and MUST be used.

Table 35: Valid Values for the TAG-byte

<table>
<thead>
<tr>
<th>TAG-byte</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Token identifier</td>
</tr>
<tr>
<td>0x01</td>
<td>Sequence number</td>
</tr>
<tr>
<td>0x02</td>
<td>Time stamp for token generation time</td>
</tr>
<tr>
<td>0x03</td>
<td>Time stamp for token expiration time</td>
</tr>
<tr>
<td>0x04</td>
<td>DN of initiator (client)</td>
</tr>
<tr>
<td>0x05</td>
<td>DN of responder (server)</td>
</tr>
<tr>
<td>0x06</td>
<td>IP address of initiator (client)</td>
</tr>
<tr>
<td>0x07</td>
<td>IP address of responder (server)</td>
</tr>
<tr>
<td>0x08</td>
<td>MAC of initiator (client)</td>
</tr>
<tr>
<td>0x09</td>
<td>MAC of responder (server)</td>
</tr>
<tr>
<td>0x0a</td>
<td>Role indicator (initiator/responder)</td>
</tr>
<tr>
<td>0x0b</td>
<td>State indicator (request/invitation)</td>
</tr>
<tr>
<td>0x0c</td>
<td>Random number 1</td>
</tr>
<tr>
<td>0x0d</td>
<td>Random number 2</td>
</tr>
<tr>
<td>0x0e</td>
<td>Random number 3</td>
</tr>
<tr>
<td>0x0f</td>
<td>Authentication mechanism, command or reply code</td>
</tr>
<tr>
<td>0x10</td>
<td>Hash value for NRR</td>
</tr>
<tr>
<td>0x11</td>
<td>Digital signature</td>
</tr>
</tbody>
</table>
According to the “Standard Guide for EDI (HL7) Communication Security” and section 9.6 of this document, all token bytes (all fields including TLV encoding) MUST be Base64-encoded, canonicalised before delivery for interoperability reasons and decoded on the server before evaluation. Base64-encoding protects against loss of data bits in environments not capable of full binary transport. Canonicalisation is performed after the encoding process to prevent system dependency. Applying neither encoding nor canonicalisation may lead to invalidation of the digital signature.

The commands and reply codes for FTP authentication MUST be implemented in the style of [26]. AUTH is used for authentication request and security mechanism transmission. ADAT is applied for transmission of authentication data. The AUTH command is used by the client to request authentication by giving an authentication mechanism as argument. Valid mechanisms must be registered with the IANA (Internet Assigned Numbers Authority) and can be found at [25] or [9]. For local use, the values begin with "X-", so for this protocol "X-SFTP" is applied.

The authentication mechanism "X-SFTP" is embedded in the token field AUTH-Mechanism of the token AuthReq', which is built according to section 9.6.2.2 step 1. All tokens containing authentication data as AuthReq, AuthData1, AuthData2 and AuthData3 are send to the server as an argument of the ADAT command. Figure 32 shows the flow of authentication tokens for SFTP.

\[
\begin{align*}
\text{SFTP/C} & \quad \text{AUTH<SPACE>AuthReq} \\
334<\text{SPACE}>\text{ADAT=AuthData1} & \quad \text{ADAT<SPACE>AuthData2} \\
\quad & \quad 235<\text{SPACE}>\text{ADAT=AuthData3}
\end{align*}
\]

\textbf{Figure 32: Flow of Authentication Tokens Exchanged for SFTP}

After the authentication has been successfully performed, authorisation based upon the user’s identity MAY be carried out by the server. The identity involved is obtained from the DN contained in the authentication tokens. Thus, no additional USER command must be used as explained in [26].

The checking of time stamps, as mentioned above, only applies either when synchronised clocks are available in a local environment, or if clocks are logically synchronised by bilateral agreements. In any case, Co-ordinated Universal Time (UTC) and secure time servers must be used.

9.7.3 Securing the Control Connection

When authenticated successfully, the control connection MUST be secured as described in section 9.6.3. The client commands and server reply codes MUST be in the style of [26]. Command tokens MUST be generated according to section 9.6.3.2 and are sent as an
argument of the security commands of [26] (for example: MIC<SPACE>Token₅ for signed transmission). The reply of the server MUST be generated analogously and the codes follow [26] (for example: 631<SPACE>Token₆).

9.7.4 Securing the Data Connection

The data connection MUST be secured as described in section 9.6.4 and provide integrity, confidentiality and non-repudiation of origin and receipt. Switching between the cryptographic protocols (e.g. S/MIME version 2, S/MIME version 3, MOSS) and their operation modes (e.g. signed-only, signed-and-encrypted) as well as selection of the session key MUST be possible. The ‘ PROT ’ command, as defined in [26], is restricted and not well specified and does not allow more than one different protocol. Therefore, this protocol uses the ‘ PROT ’ command with a word encoded argument (2 Bytes in little endian order).

The first byte (low byte) MUST state the cryptographic protocol and its operation modes as detailed in Table 36. All unused entries of this byte between hexadecimal 0x00 and 0x3F are user-definable, other values MUST not be allocated or re-allocated.

For now, MIME Security Auto-detection (value 0x3F) MUST be used only for MIME-object security protocols, such as Security Multi-parts For MIME, S/MIME version 2 and 3, MOSS and PGP/MIME. When setting auto-detection, the receiving application knows that something is transmitted using MIME-object security, but it neither knows the specific MIME-object security protocol nor the operation mode: signed-only, encrypted-only or signed-and-encrypted. The auto-detection mechanism identifies the MIME-object security protocol and the operation modes. Furthermore, this mechanism must be able to process files containing multiple messages that may also vary in their MIME-object security protocol and operation mode. Automatic detection is based upon the MIME type indicated by the content-type and the evaluation of accompanying parameters.

Table 36: Encoding for the Cryptographic Protocol and its Operation Mode

<table>
<thead>
<tr>
<th>Value</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Plain Text (ASCII)</td>
</tr>
<tr>
<td>0x10</td>
<td>PKCS#7-only</td>
</tr>
<tr>
<td>0x11</td>
<td>CMS-only</td>
</tr>
<tr>
<td>0x20</td>
<td>Security Multi-parts For MIME</td>
</tr>
<tr>
<td>0x21</td>
<td>S/MIME Version 2</td>
</tr>
<tr>
<td>0x22</td>
<td>S/MIME Version 3</td>
</tr>
<tr>
<td>0x30</td>
<td>MOSS</td>
</tr>
<tr>
<td>0x31</td>
<td>PGP/MIME</td>
</tr>
<tr>
<td>0x3F</td>
<td>MIME Security Auto-detection</td>
</tr>
<tr>
<td></td>
<td><strong>Operation Mode:</strong></td>
</tr>
<tr>
<td>0x40</td>
<td>Sign</td>
</tr>
<tr>
<td>0x80</td>
<td>Encrypt</td>
</tr>
</tbody>
</table>
The operation modes SHOULD only be given if non-MIME protocols are used as PKCS#7-only and CMS-only. In this case, the value stating the protocol and the value of the desired operation modes are combined using OR-operations bit by bit. For example, PKCS#7-only in signed-and-encrypted operation mode will result in the value \((0x10 \text{ OR } 0x40) \text{ OR } 0x80) = 0xD0\).

The second byte (high byte) of the ‘PROT’-command argument MUST define the session key algorithm as shown in Table 37. Here, all unused entries are user-definable.

<table>
<thead>
<tr>
<th>Value</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>IDEA</td>
</tr>
<tr>
<td>0x10</td>
<td>DES-EDE2-CBC</td>
</tr>
<tr>
<td>0x11</td>
<td>DES-EDE3-CBC</td>
</tr>
</tbody>
</table>

### 9.7.5 Security Considerations Regarding the Protocol Stack

According to the “Standard Guide for EDI (HL7) Communication Security,” the specification of the protocols used, such as FTP, TCP, and IP contains a number of mechanisms that can be used to compromise network security. There are many known Internet attacks based on infrastructure weakness, such as DNS spoofing, source routing (IP spoofing), FTP bouncing, racing authentication and denial of service. Attacks arising from the weakness of the FTP protocol and underlying protocols SHOULD be addressed by this protocol regarding [6] or 1.

Racing authentication, which is based on faster authentication of the attacker than the victim, SHOULD be prevented by the strong mutual three-way authentication, based on challenge/response and digital signature, and the restriction to one simultaneous login of the same user. Moreover, the total number of control connection possible SHOULD also be limited.

To protect against FTP bouncing (namely the misuse of the PORT command), the server SHOULD not establish connections to arbitrary machines (for instance a second FTP server called proxy FTP) and ports on these machines. Following 1 and [6], the server SHOULD ensure that the IP address specified in the PORT command matches the client’s source IP address for the control connection. Furthermore, the server MUST disallow data connections if the TCP-port specified in the PORT command is a well-known port (port 0 to 1023) or registered port (1024 to 49151). Only dynamic, private ports (port 49152 to 65535) are allowed. Hence, a port scan against another site hiding the true source and bypassing access controls like firewalls cannot be performed (for instance bouncing to a well-known port). The PORT command is used in the active mode only It is not used in the passive mode that is initiated by the PASV command. Since the PASV command is not affected by the bounce attack since the server gives the IP address and port to connect to and an attacker cannot act as a server, it is preferred to the PORT command providing firewall-friendly FTP (see [17]) as well. Using passive initiation of the data connection means that the TCP connection establishment is performed from the client network toward the server network.
Furthermore, random local port numbers SHOULD be used for the data connection, as stated in [6], to address port number guessing. Guessing the next port number is much easier when simple, increasing algorithms are used (for example: next port = old port + constant number). Using simple, increasing algorithms enables attacks like the denial of a data connection or the hijacking a data connection to steal files or insert forged files.

In addition to the authentication procedures, access restrictions based on network addresses MAY be provided. The server accepts only connection requests from pre-defined IP addresses within authorised organisations and confirms this address matches on both the control connection and the data connection. When relying on IP address authentication only, an attack like source routing of IP packets (IP spoofing) is possible.

To address DNS spoofing, hostname to IP address resolution or vice versa (DNS) SHOULD NOT be performed by client or server. The destination machine SHOULD be reached by the IP address directly.

For the detection of compromises such as denial of service attacks and other attacks), the server SHOULD keep reports logging all activities including connection attempts, disconnection, command executions and others. Since local machines are considered trusted, integrity and/or confidentiality protection is not required.

9.8 Example For PKCS#7-based Security (informative)

For application of PKCS#7-only security, the plain data file is available on the file system. Next, this file is signed, applying the signedData object of PKCS#7 with the contentInfo field carrying the message data. At last, this object is encrypted using the envelopedData object of PKCS#7. After transportation, this file is processed conversely (decryption following verification)

9.9 Example For Security Multipart for MIME (informative)

In this subsection an example is presented, in which an HL7 message is secured by hybrid encryption using Security Multi-parts for MIME, as specified in [19] and PKCS#7, for signing and encryption. For hybrid data encryption, a nesting of content-types is performed as explained in [12, 29] using a triple DES session key (112 bits significant, DES3-EDE2-CBC).

| MSH | ^~\&|DPS||CLOVERLEAF||19970922075909||ADT^A08|0165648|P|2.2||AL|NE
| EVN|A08|19970922075857
| PID|123456|SSW23084913|97045331|Sorglos^Susi||19490823||Milchstrasse
| 99``Magdeburg``39999``D|15303000|6211123||deutsch||0|||||||
| NK1|1|Sorglos^Harry|EHENANN|Milchstrasse 99``Magdeburg``39999``D|6211123
| PU1|S|MKG01``MKG|R||1||1||1||1||1||1||1||97999999||1997017084100
| DG1|1|ICD9|2398||19970917085134|AUF
| IN1|1|001441346|0969999|HaMo Ost/Gst Magdeburg|Keplerstraße
| 6``Magdeburg``39104``D||1|HAM||1200||Sorglos^Susi||19490823|Milchstrasse
| 99``Magdeburg``39999``D||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1||1|
First, the HL7 sample message as shown in Figure 33 is available on the file system in plain text.

This message is Base64-encoded and inserted into a MIME entity using the content-type "application/x-EDI-HL7" as proposed in [8] (regarding [12, 18]). For readability of the HL7 messages, the quoted-printable encoding could be implemented. Next, this entity is canonicalised as presented in Figure 34 (This means 7-bit ASCII representation with lines terminated by carriage return <CR> and line feed <LF> as specified in [20] chapter 2.1.1.)

```
Content-Type: application/x-EDI-HL7; charset=us-ascii<CR><LF>
Content-Transfer-Encoding: base64<CR><LF>
Content-Description: HL7 V2.2 message<CR><LF>

TVNIfF5+XC28RFBTfHxDT9RJRjU5FGFHwxOTk3MDkyMjA3NTkwOXx8QURXkBw<CR><LF>
OHwMTY1NjQ4fPBK14YfHx8QXix8TKUVRJzoFEEwOHwxOTk3MDkyMjA3NTgwNwpQ<CR><LF>
SUR8fDEyMzQ1NnxTUlcyWzA4NDkwM3wSNzA0NTMzMXxTb3JmG9X1Nlc218fDB5<CR><LF>
NDkwOZhzziF8fHxNaWxjaHNCmFzc2UgOlTeXk1hZ2R1YnVz15eMzk5OTleRHz<CR><LF>
NTMwMzAWHw2MjExMTlzfHxkZXVoC2NoehHwwfHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8<CR>
NDk5XkR8NjIxMTk8YnpQVjF8fFN8TUt1MDPhxL5NS0d8Unx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8<CR><LF>
U3w5Nzk5OTkxQ8f8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8<CR><LF>
MDg0MTAwCkRJv0fHwxFEBDR18MjMSOHx8MTk5NzA5MzOWJ2dUxMzR8QVVGck1OMXw<CR><LF>
DwMTQOMM0nwwOTY5OTk5fEhhtFtwgTgcgT1NOL0dzdCBNYWdkZWJlcmd8S2Vw<CR><LF>
bGvyc3lyYd91lDZXkh1x22Z1YnVz15eMzk5MDReRHzfDP88EFNfHx8fDY5MB8<CR><LF>
fE18U29yZ2xvc15tDxNgfHwxOTQ5MDgyM3wNaWxjAHNcmFzc2Vexk1hZ2RlyNvY<CR><LF>
Z15eMzk5OTleRHz8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8<CR><LF>
```

**Figure 34: MIME Entity of the HL7 Sample Message**

Then, the MIME entity given in Figure 34 is signed using PKCS#7. Following [19], the MIME entity is inserted into a multipart/signed MIME message representing the first body part as depicted in Figure 35. The digital signature (PKCS#7-signedData object with an empty contentInfo field as explained in [29]) is Base64-encoded and inserted into the second body part. According to [19], the signature covers the MIME header of the first body and the first body itself as marked bold in Figure 35.
Content-Type: multipart/signed; <CR><LF>
    protocol = "application/pkcs7-signature"; <CR><LF>
    micalg = md5; boundary = sig_bound<CR><LF>
<CR><LF>
--sig_bound-- <CR><LF>

Content-Type: application/x-EDI-HL7; charset=us-ascii<CR><LF>
Content-Transfer-Encoding: base64<CR><LF>
Content-Description: HL7 V2.2 message<CR><LF>
<CR><LF>
TVNIfF5+XC28RFBTfHxDTE9WRVJMRUFGfHwxOTk3MDkyMjA3NTkwOXx8QURUXkEw<CR><LF>
OHwxMTY1NjQ4fFB8Mi4yfHx8QUX8TkUKR5V0fEEOBHwxOTk3MDkyMjA3NTgwQ<CR><LF>
SUR8fDEyMzQ1NnxTULVcyMzA4NDkxM3w5NzA0NTMzMXxtB3JnbG9zX1n1c2l8fDE5<CR><LF>
NDkODIZfFpd8fHxNaWxjaHNN0cmFsc2UgOT1eKlhlZ2RlYnVyZ15eMzk5OT1eRWx<CR><LF>
NTMwMzAxMHRw2MjExMTIzFHxkZXV0c2NoFHw0FHx8FHx8FHx8RApOSzF8MxxtB3Jn<CR><LF>
bG9xRkhycnJ5fEVIU1BTk58TW1sV2hzdHJc3N1IDkX15NYWdk WorceXjMY<CR><LF>
OTk5XR8NJxMTMByHwQfV8fFN8TUTHMDEx15NS04UHx8FHx8FHx8FHx8FHx8FHx8<CR><LF>
U3w5NzaxOTk5OXx8FHx8FHx8FHx8FHx8FHx8FHx8FHx8FHx8FHx8FHx8FHx8FHx8FHx8<CR><LF>
MDg0MTAwCkRHRMxWxfelDR18Mj50OHx4MT5x5Az5MCw0ODUxMzR8QVVGc3I0Mxwx<CR><LF>
fDAwMTQ0MTM0NnwwOTY5OTk5F5ehHTWwFgTcrg3NOL0dzDCBNTWdKZW1jcmd8S2Vw<CR><LF>
bGVyc3Ry9Y9DZexZkliZ2R1YnVyZ15eMzkxMDReRHx8fDF8fSEfN8FHx8fDEyMDB8<CR><LF>
fB18U29y9Zxvxc15tDNfHwy0fQ5MDgyM3xNaWxjaHN0cmFzc2VxkZ2RlYnVy<CR><LF>
215eMzk5OT1eRHx8FHx8FHx8FHx8fHw0fOT5OTk5Xjk5NjA0XjEyMDBeMTAw<CR><LF>
MP45Cg==<CR><LF>
--sig_bound-- <CR><LF>

Content-Type: application/pkcs7-signature; name=smime.p7s<CR><LF>
Content-Transfer-Encoding: base64<CR><LF>
Content-Disposition: attachment; filename=smime.p7s<CR><LF>

MIAGCSqGSIb3DQEHAqCAMIACAQExdjAMBggqkhkJwG0CBQUAMICAGScqGSIb3DQEHE<CR><LF>
AQAAMYGHiMAEgg5oEMLAAGA1uUEBsbMCReUUDDAkbGnvAeTA2dtZAIIBD<CR><LF>
BggqkhkJG0OCBQUA0GCSqGSIb3DQUEBAdUEBBAUDEUT2sgc2Vz0HmNfeifA37c7dW<CR><LF>
Kw7HUeQKdb8QXXfI10VBMYPR4eKX0vmQfmpqWhSkSpT4eLdCFzSoSvAAAA<CR><LF>
AAA<CR><LF>
--sig_bound-- <CR><LF>

Figure 35: Signed HL7 Sample Message Using Secure MIME Multiparts

Next, the whole multipart/signed message (including MIME headers and trailers, as presented in Figure 35, is encrypted using the PKCS#7-envelopedData object. The result is shown in Figure 36. According to [19] the first body parts contain control information (control value and protocol parameter) to decrypt the data in the second body part. After transportation, the HL7 message is decrypted and verified using an analogue method. For validation of the digital signature, the part covered by the signature must be canonicalised first.
Chapter 9

Content-Type: multipart/encrypted; boundary=enc_bound

--enc_bound

Content-Type: application/pkcs7-mime; charset=us-ascii

Content-Description: control information for decryption

Version: 1.5

Content-Type: application/octet-stream; name=smime.p7m

Content-Description: secure MIME (RFC1847) cryptographic message

MIAGCSqGSIb3DQEHA6CAMIACAQAxDB2AgEAMCAwGzELMAkGAlUEBhMCREUxDDAK

BgNVBAoTA2dtZAdANBgkqhkiG9w0BAQsFAAOCAQDp+/hxWniud4Ifd30pcn<CR><LF>

Oywere8x14Crr0Z7e+q6eVBMYNo2r9LXC0i14B9YuVFQoKTG9AwUn20OEiTATCA<CR><LF>

Bkgqhkl9j09OBWbEwCQYFKQDAQ provisional A2CBIIG8Fd/b/UBg8KZb29Jfg/E1X4di22<CR><LF>

Cry50MC99f7Q0zUNu6jeXLm8Ed/hI9RI2Mda4Y4XcGWhGy3wpF<CR><LF>

NFaqWmNh4walR7vRkRvFkqyZPc1cZNYDlYT79KF/ZFVw5a2Vp+jiP0J21A8JzooACr<CR><LF>

mlS2YP8GvljDCyUPTd7d7y5ztkAVZCC4sYHMLtY9fes7t7ia7rQzuFBPGgqtHFh<CR><LF>

jIEkXkVrVunjmeEq1dVcmSuhefJ7c4+BpkywPM7qilo75QM/TenY62sgmuAXCS<CR><LF>

FMHRkRRyHdyRr9MwGQVR/VRhsZ82Og8eMQe289APOo+NC7dilQYGR0F+Gj0yTocr<CR><LF>

TLo9j1HZmWtONDGQ0rWi4hunLrccdWAKzHaa5C0I0Y9hNLDlyf ELFZHBFrrrek9<CR><LF>

G1xDhltI5gNBj8rSilSAGzRAAZz+VFSNtE4WEdRsLuM0UPCzm1pb7PJoieix+tL+Ei<CR><LF>

A7jTZ93cOOGHsDitDYN76k/b40XgluelYg19F3x+yXvNBAHlYBE6B+1r9fz3TUr<CR><LF>

lu6Hutl72oHos0n3WdoTjfcqRMp7M4rnmijfUBfShsVdcCFACX4u6PUiRllc<CR><LF>

gu4enklpy/7qycR8ja5h9eB6gQf2+1f13Gp99cFgwG3/pToP4UrVY2<CR><LF>

UY8HZc3bouMeDJ3FC5AU3qKgMFCK+5Xm6h6T5y5U/qtnHStYbgNT/6Fpwe/nL<CR><LF>

fyjyQ/yC9UDAXJlM1KmHEH0poG0pXOaw6YpWvnHJ0u8g+wX26XH0koHae6a5Q<CR><LF>

XzrR1Q4iK2d1Lza0R0X5sgeCP2fuD5e12s/LKbEPw+ozgS701NrVv2zCAhCJGy9iu<CR><LF>

VG4NeRLCptUKJkFw2dLXYXQJbh15Ko6jNFyKAFrgAcAgzh83b3x1ZmaAFAO/1<CR><LF>

Kylal8kI5wULvSnmNga32E6CRX4E5pUk3i01VS0hj6W5ac16G5iQn2d/gzFNeq<CR><LF>

+OBojQ4kL80XLbhGqGcmv4/jf1WhryDENLdXmhrctGyuY/1Y58PfHeNCayf<CR><LF>

cE4GuAc81A18s0lLtZ9DNU1NbhI0ROZ+wikbMT1AymVRL1kJt/Bj1CPBwvd1xECQ2<CR><LF>

IplQb9hutXdbnrshpUOpizVNnjDk+7C8S5F/k1d1DKM4W8nFmQnzWuiJc8mU<CR><LF>

ILyAPvXvKHFUtztLQye+y5BbMKEm5Wk0Pe1dFPG6YtxcmDyeIWP65aytHYBefm2a<CR><LF>

hhqOPqjCprHHzCRyYtDxOBdDCg2y2vhp6Z9e70XlBPVR90ze6u+2+GzN6iku<CR><LF>

U12pbovVRMplEq1qdIzXq5y089fLg0ux4waA3VypBwQWlhxK7rb4DBKN10P<CR><LF>

C7aouuem6QnmpjO2c<hg/dbB86kK7nkdf0uhr3biC8ZfdRfGzhyhDeAcORrn0Aq<CR><LF>

mja0ZGFknacZudObmoSW9kUsPv/N1B4U7Qos6FSSBv+odYJuBi381AquatB<CR><LF>

md<CR><LF>

xD5k0HTVp8KPGF50F2F2pX7n4mWvUWfsc/CkYUyGrDFvWvN5BqiaZEKeWacz+<CR><LF>

0jS4PMffVawNlTyZihm58G9T/A+iW+KRYDh3trvVZ2585D5P7NFpTzCoxYv8L+wfe<CR><LF>
For signed-only transportation, the process ends up in the multipart/signed-structure as presented in Figure 36. In the case of encrypted-only delivery, the MIME entity in Figure 35 is converted to the multipart/encrypted-structure directly without any intermediate steps.

9.10 Example For S/MIME Version 2 (Informative)

In this subsection, an example is presented, in which an HL7 message is secured by hybrid encryption, applying DES3-EDE2-CBC, using S/MIME version 2 as specified in [29] and PKCS#7 for signing and encryption. The HL7 sample message as shown in Figure 34 is available on the file system in plain text.

This message is Base64-encoded and inserted into a MIME entity using the content-type "application/x-EDI-HL7" that is canonicalised afterwards as presented in Figure 35. Next, this entity is signed using the PKCS#7-signedData object with the ContentInfo field carrying the MIME entity) as explained in [29]. Alternatively, multipart/signed can be used for signing. At last, the PKCS#7 object is inserted into an application/pkcs7-mime MIME entity as shown in Figure 37.

For encryption, the signed message given in Figure 37 is enveloped using the PKCS#7-envelopedData object and inserted into an application/pkcs7-mime MIME entity as described in [29] and shown in Figure 38.
Chapter 9

Figure 37: Signed HL7 Sample Message Using S/MIME Version 2

After transportation the HL7 message is decrypted and verified using an analogue method. For validation of the digital signature, the part covered by the signature must be canonicalised first.
For signed-only transportation, the process ends up in the application/pkcs7-mime-structure as presented in Figure 37. In the case of encrypted-only delivery, the MIME entity in Figure 35 is converted to the structure represented in Figure 38 directly without any intermediate step.

9.11 References and Bibliography (Informative)


   http://www.cert.org/nav/securityimprovement.html,
   ftp://ftp.cert.org/pub/tech_tips/FTP_PORT_attacks,
   http://www.cert.org/advisories/CA-97.27.FTP_bounce.html,
   http://www.cert.org/advisories/
   CA-95.01.IP.spoofing.attacks.and.hijacked.terminal.connections.html).


9 Internet Assigned Numbers Authority (IANA): Directory of General


13 Multipurpose Internet Mail Extensions (MIME) Part One-Five, Request for Comments: 2045, 2046, 2047, 2048, 2049 (Network Working Group), N. Freed, N. Borenstein (2045, 2046, 2049); K. Moore (2047); N. Freed, J. Klensin, J. Postel (2048), November 1996. ftp://ftp.isi.edu/in-notes.


Also available as (to replace RFC2112):


Also available as:

29 S/MIME Version 2, Request for Comments (Network Working Group):
ftp://ftp.isi.edu/in-notes,

30 S/MIME Version 3, Internet Draft (S/MIME Working Group):
http://www.ietf.org/html.charters/smime-charter.html,

Also available as ISO/IEC9594-8.