Open Financial Exchange Specification 1.0.2

May 30, 1997

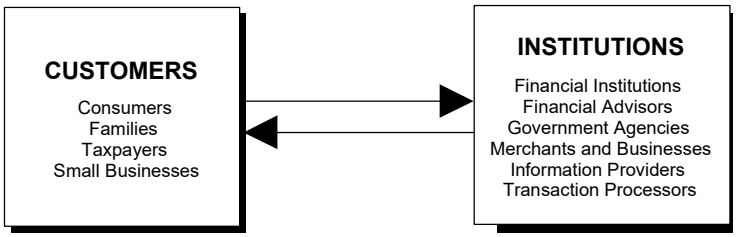
Chapters 1 – 4

Contents

1. Overview

1.1 Introduction

Open Financial Exchange is a broad-based framework for exchanging financial data and instructions between customers and their financial institutions. It allows institutions to connect directly to their customers without requiring an intermediary.



Open Financial Exchange is an open specification that anyone can implement: any financial institution, transaction processor, software developer, or other party. It uses widely accepted open standards for data formatting (such as SGML), connectivity (such as TCP/IP and HTTP), and security (such as SSL).

Open Financial Exchange defines the request and response messages used by each financial service as well as the common framework and infrastructure to support the communication of those messages. This specification does not describe any specific product implementation.

1.1.1 Design Principles

The following principles were used in designing Open Financial Exchange:

- **Broad Range of Financial Activities** – Open Financial Exchange provides support for a ***broad*** range of financial activities. Open Financial Exchange 1.0.1 specifies the following services:
  - Bank statement download
  - Credit card statement download
  - Funds transfers including recurring transfers
  - Consumer payments, including recurring payments
  - Business payments, including recurring payments
  - Brokerage and mutual fund statement download, including transaction history, current holdings, and balances.

- **Broad Range of Financial Institutions** – Open Financial Exchange supports communication with a ***broad*** range of financial institutions (FIs), including:
  - Banks
  - Brokerage houses
  - Merchants
  - Processors
  - Financial advisors
  - Government agencies
- **Broad Range of Front-End Applications** – Open Financial Exchange supports a ***broad*** range of front-end applications, including Web-based applications, covering all types of financial activities running on all types of platforms.
- **Extensible** – Open Financial Exchange has been designed to allow the easy addition of new services. Future versions will include support for many new services.
- **Open** – This specification is publicly available. You can build client and server applications using the Open Financial Exchange protocols independent of any specific technology, product, or company.
- **Multiple Client Support** – Open Financial Exchange allows a user to use multiple client applications to access the same data at a financial institution. With the popularity of the World Wide Web, customers are increasingly more likely to use multiple applications—either desktop-based or Web-based—to perform financial activities. For example, a customer can track personal finances at home with a desktop application and occasionally pay bills while at work with a Web-based application. The use of data synchronization to support multiple clients is a key innovation in Open Financial Exchange.
- **Robust** – Open Financial Exchange will be used for executing important financial transactions and for communicating important financial information. Assuring users that transactions are executed and information is correct is crucial. Open Financial Exchange provides robust protocols for error recovery.
- **Secure** – Open Financial Exchange provides a framework for building secure online financial services. In Open Financial Exchange, security encompasses authentication of the parties involved, as well as secrecy and integrity of the information being exchanged.
- **Batch & Interactive** – The design of request and response messages in Open Financial Exchange is for use in either batch or interactive style of communication. Open Financial Exchange provides for applying a single authentication context to multiple requests in order to reduce the overhead of user authentication.
- **International Support** – Open Financial Exchange is designed to supply financial services throughout the world. It supports multiple currencies, country-specific extensions, and different forms of encoding such as UNICODE.
- **Platform Independent** –Open Financial Exchange can be implemented on a wide variety of front-end client devices, including those running Windows 3.1, Windows 95, Windows NT, Macintosh, or UNIX. It also supports a wide variety of Web-based environments, including those using HTML, Java, JavaScript, or ActiveX. Similarly on the back-end, Open Financial Exchange can be implemented on a wide variety of server systems, including those running UNIX, Windows NT, or OS/2.
- **Transport Independent** – Open Financial Exchange is independent of the data communication protocol used to transport the messages between the client and server computers. Open Financial Exchange 1.0.1 will use HTTP.

1.2 Open Financial Exchange at a Glance

The design of Open Financial Exchange is as a client and server system. An end-user uses a client application to communicate with a server at a financial institution. The form of communication is requests from the client to the server and responses from the server back to the client.

Open Financial Exchange uses the Internet Protocol (IP) suite to provide the communication channel between a client and a server. IP protocols are the foundation of the public Internet and a private network can also use them.

1.2.1 Data Transport

Clients use the HyperText Transport Protocol (HTTP) to communicate to an Open Financial Exchange server. The World Wide Web throughout uses the same HTTP protocol. In principle, a financial institution can use any off-the-shelf web server to implement its support for Open Financial Exchange.

To communicate by means of Open Financial Exchange over the Internet, the client must establish an Internet connection. This connection can be a dial-up Point-to-Point Protocol (PPP) connection to an Internet Service Provider (ISP) or a connection over a local area network that has a gateway to the Internet.

Clients use the HTTP POST command to send a request to the previously acquired Uniform Resource Locator (URL) for the desired financial institution. The URL presumably identifies a Common Gateway Interface (CGI) or other process on an FI server that can accept Open Financial Exchange requests and produce a response.

The POST identifies the data as being of type application/x-ofx. Use application/x-ofx as the return type as well. Fill in other fields per the HTTP 1.0 spec. Here is a typical request:

```
POST http://www.fi.com/ofx.cgi HTTP/1.0        HTTP headers
User-Agent:MyApp 5.0
Content-Type: application/x-ofx
Content-Length: 1032

OFXHEADER:100                                  OFX headers
DATA:OFXSGML
VERSION:102
SECURITY:TYPE1
ENCODING:USASCII

<OFX>                                          OFX request
... Open Financial Exchange requests ...
</OFX>
```

A blank line defines the separation between the HTTP headers and the start of the Open Financial Exchange headers. A blank line also separates the Open Financial Exchange headers and the request. (See Chapter 2 for more information about the Open Financial Exchange headers.)

The structure of a response is similar to the request, with the first line containing the standard HTTP result, as shown next. The content length is given in bytes.

```
HTTP 1.0 200 OK                                    HTTP headers
Content-Type: application/x-ofx
Content-Length: 8732

OFXHEADER:100                                      OFX headers
DATA:OFXSGML
VERSION:101
SECURITY:TYPE1
ENCODING:USASCII

<OFX>                                              OFX response
... Open Financial Exchange responses ...
</OFX>
```

1.2.2 Request and Response Model

The basis for Open Financial Exchange is the request and response model. One or more requests can be batched in a single file. This file typically includes a signon request and one or more service-specific requests. An FI server will process all of the requests and return a single response file. This batch model lends itself to Internet transport as well as other off-line transports. Both requests and responses are plain text files, formatted using a grammar based on Standard Generalized Markup Language (SGML). Open Financial Exchange is syntactically similar to HyperText Markup Language (HTML), featuring tags to identify and delimit the data. The use of a tagged data format allows Open Financial Exchange to evolve over time while continuing to support older clients and servers.

Here is a simplified example of an Open Financial Exchange request file. (This example does not show the Open Financial Exchange headers and the indentation is only for readability.) For complete details, see the more complete examples throughout this specification.

The response format follows a similar structure. Although a response such as a statement response contains all of the details of each transaction, each element is identified using tags.

The key rule of Open Financial Exchange syntax is that each tag is either an element or an aggregate. Data follows its element tag. An aggregate tag begins a compound tag sequence, which must end with a matching tag; for example, <AGGREGATE> ... </AGGREGATE>.

The file sent by Open Financial Exchange does not contain any white space between tags.

1.3 Conventions

The conventions used in the tag descriptions include the following:

- Required tags are in **bold**. Regular face indicates tags that are optional. Required means that a client must always include the tag in a request, and a server must always include the tag in a response.
- Required tags occur once unless noted as one or more in the description, in which case the specification allows multiple occurrences.
- Optional tags occur once if present unless noted as zero or more in the description, in which case the specification allows multiple occurrences.

- A-*n* or N-*n* specify those values that take an alphanumeric or numeric type value, where *n* indicates the maximum size.
- Common value types, such as a dollar amount, are referenced by name. Chapter 3 lists value types that are referenced by name.

Tag	*Description*
<REQUIREDTAG>	Required tag (1 or more)
<REQUIREDTAG2>	Required tag that occurs only once
<OPTIONALTAG>	Optional tag; this tag can occur multiple times (0 or more)
<SPECIFIC>	Values are A, B, and C
<ALPHAVALUE>	Takes an alphanumeric value up to 32 characters, *A-32*

2. Structure

This chapter describes the basic structure of an Open Financial Exchange request and response. Structure includes headers, basic syntax, and the Signon request and response. This chapter also describes how Open Financial Exchange encodes external data, such as bit maps.

Open Financial Exchange data consists of some headers plus one Open Financial Exchange data block. This block consists of a signon message and zero or more additional messages. When sent over the Internet using HTTP, standard HTTP and multi-part MIME headers and formats surround the Open Financial Exchange data. A simple file that contained only Open Financial Exchange data would have the following form:

```
HTTP headers
MIME type application/x-ofx
Open Financial Exchange headers
Open Financial Exchange SGML block
```

A more complex file that contained additional Open Financial Exchange data would have this form:

```
HTTP headers
MIME type multipart/x-mixed-replace; boundary =--boundary—
---boundary---
MIME type application/x-ofx
     Open Financial Exchange headers
     Open Financial Exchange SGML block

---boundary---
   MIME type image/jpeg
      FI logo
```

2.1 HTTP Headers

Data delivered by way of HTTP places the standard HTTP result code on the first line. HTTP defines a number of status codes. Servers can return any standard HTTP result. However, FIs should expect clients to collapse these codes into the following three cases:

Code	*Meaning*	*Action*
200	OK	The request was processed and a valid Open Financial Exchange result is returned.
400s	Bad request	The request was invalid and was not processed. Clients will report an internal error to the user.
500s	Server error	The server is unavailable. Clients should advise the user to retry shortly.

NOTE: *The server must return code 400 for any problem that prevents it from processing the request file. Processing problems include failures relating to security, communication, parsing, or the Open Financial Exchange headers (for example, the client requested an unsupported language). For content errors such as wrong USERPASS or invalid account, the server must return a valid Open Financial Exchange response along with code 200. If a communication time-out error occurs while an OFX server and a back-end server are communicating to fill a request, then the server MUST return code 500.*

Open Financial Exchange requires the following HTTP standard headers:

Code	*Value*	*Explanation*
Content-type	application/x-ofx	The MIME type for Open Financial Exchange
Content-length	length	Length of the data after removing HTTP headers

When responding with multi-part MIME, the main type will be multi-part/x-mixed-replace; one of the parts will use application/x-ofx.

2.2 Open Financial Exchange Headers

The contents of an Open Financial Exchange file consist of a simple set of headers followed by contents defined by that header

The Open Financial Exchange headers are in a simple *tag:value* syntax and terminated by a blank line. Open Financial Exchange always sends headers unencrypted, even if application-level encryption is used for the remaining contents. The language and character set used for the headers is the same as the preceding MIME headers.

The first entry will always be OFXHEADER with a version number. This entry identifies the contents as an Open Financial Exchange file and provides the version number of the Open Financial Exchange headers that follow (not the version number of the contents). For example:

```
OFXHEADER:100
```

Open Financial Exchange headers can contain the following tags.

```
DATA:OFXSGML
VERSION:102
SECURITY:
ENCODING:
CHARSET:
COMPRESSION:
OLDFILEUID:
NEWFILEUID:
```

A blank line follows the last header. Then (for type OFXSGML), the SGML-readable data begins with the <OFX> tag.

For information about the OFX headers, refer to the following sections.

2.2.1 OFXHEADER

OFXHEADER specifies the version number of the Open Financial Exchange headers.

The OFXHEADER value should change its major number only if an existing client is unable to process the new header. This can occur because of a complete syntax change in a header, or a significant change in the semantics of an existing tag—not the entire response. A server can add new tags as long as clients can function without understanding them.

The current version of the Open Financial Exchange headers is version 1.0 (OFHEADER:100).

2.2.2 DATA

DATA specifies the content type, in this case OFXSGML.

You should add new values for a DATA tag only when you introduce an entirely new syntax. In the case of OFXSGML, a new syntax would have to be non-SGML compliant to warrant a new DATA value. It is possible that there will be more than one syntax in use at the same time to meet different needs.

2.2.3 VERSION

VERSION specifies the version number of the Document Type Definition (DTD) used for parsing. The current version of the DTDs is version 1.0.2 (VERSION:102).

The VERSION tag identifies syntactic changes. In the case of OFXSGML, this corresponds to the DTD. Purely for identification purposes, each change increments the minor number of the version tag. If you introduce an incompatible change so that an older DTD can not parse the file, the major number should change. See the general discussion of message sets and version control, later in this chapter.

> ***NOTE:*** *VERSION provides the version number of the DTD. The <OFX> block describes the version numbers of specific message sets.*

2.2.4 SECURITY

SECURITY defines the type of application-level security, if any, that is used for the <OFX> block. The values for SECURITY can be NONE or TYPE1.

For more information about security, refer to Chapter 4, “Security.”

2.2.5 ENCODING and CHARSET

ENCODING defines the text encoding used for character data. The values for ENCODING can be UNICODE or USASCII.

CHARSET defines the character set used for character data.

For more information about ENCODING and CHARSET, refer to Chapter 5, “International Support.”

2.2.6 COMPRESSION

A future version of the specification will define compression.

2.2.7 OLDFILEUID and NEWFILEUID

NEWFILEUID uniquely identifies this request file. The NEWFILEUID, which clients must send with every request file and which servers must echo in the response, serves several purposes:

- Servers can use the NEWFILEUID to quickly identify duplicate request files.
- Clients and servers can use NEWFILEUID in conjunction with OLDFILEUID for file-based error recovery. For more information about using file-based error recovery or *lite synchronization*, see Chapter 6, "Data Synchronization."
- Servers can use the NEWFILEUID to manage the session keys associated with Type 1 application-level security. For more information about security, refer to Chapter 4, "Security."

OLDFILEUID is used together with NEWFILEUID only when the client and server support file-based error recovery. OLDFILEUID identifies the last request and response that was received and processed by the client.

2.3 SGML Details

2.3.1 Compliance

SGML is the basis for Open Financial Exchange. A DTD formally defines the SGML wire format for Open Financial Exchange. However, Open Financial Exchange is not completely SGML-*compliant* because the specification allows unrecognized tags to be present. Clients and servers must skip over the unrecognized tags. That is, if a client or server does not recognize <XYZ>, it must ignore the tag and its enclosed data. A fully-compliant SGML parser would not *validate* a document that contains tags that the DTD does not define.

Although SGML is the basis for the specification, and the specification is largely compliant with SGML, do not assume Open Financial Exchange supports any SGML features not documented in this specification.

2.3.2 Valid SGML Characters

Open Financial Exchange tags that require a value can be set to any sequence of SGML characters. To be valid, a value must contain at least one character that is not a blank character. In other words, a value cannot contain only white space.

2.3.2.1 Special Characters

For the purposes of Open Financial Exchange, a few characters must be handled as special characters. To represent a special character, use the corresponding escape sequence.

Character	*Escape sequence*
< (less than)	&lt;
> (greater than)	&gt;
& (ampersand)	&amp;

For example, the string "AT&T" encodes "AT&T."

NOTE: *Escape sequences are not required when these special characters are used in tag values that accept HTML-formatted strings (for instance, e-mail records). These tags accept SGML-marked section syntax that hides the HTML from the Open Financial Exchange parser. You must prefix the HTML-formatted strings with "<![CDATA]"and suffix them with "]]>." Within these bounds, treat the above characters literally without an escape. See Chapter 9 for an example.*

2.3.3 Comments Not Supported

For explanatory purposes, the examples in this specification contain comments. However, Open Financial Exchange files *cannot* contain comments.

2.4 Open Financial Exchange SGML Structure

2.4.1 Overview

Open Financial Exchange hierarchically organizes request and response blocks:

Top Level <OFX>
Message Set and Version <*XXX*MSGSVn>
Synchronization Wrappers <YYYSYNCRQ>, <YYYSYNCRS>
Transaction Wrappers <YYYTRNRQ>, <YYYTRNRS>
Specific requests and responses

The following sections describe these levels.

2.4.2 Case Sensitivity

OFX requires upper case letters for element names and enumerated values. In the example below, <SEVERITY> is an element with an enumerated value and <MESSAGE> is an element with a value that is not enumerated.

```
<STATUS>
        <CODE> 2000
        <SEVERITY> ERROR
        <MESSAGE> General Error
</STATUS>
```

2.4.3 Top Level

An Open Financial Exchange request or response has the following top-level form:

Tag	*Description*
<OFX>	Opening tag
... Open Financial Exchange requests or responses ...	0 or more transaction requests and responses inside appropriate message set aggregates
</OFX>	Closing tag for the Open Financial Exchange record

This chapter specifies the order of requests and responses.

A single file MUST contain only one OFX block.

2.4.4 Messages

A message is the unit of work in Open Financial Exchange. It refers to a request and response pair, and the status codes associated with that response. For example, the message to download a bank statement consists of the request <STMTRQ> and the response <STMTRS>. In addition, with the exception of the signon message, each message includes a *transaction wrapper*. For requests, the transaction wrapper adds a transaction unique ID <TRNUID>. For responses, the transaction wrapper adds the same transaction unique ID <TRNUID>, plus a <STATUS> aggregate.

For messages subject to synchronization (see Chapter 6), a third layer of aggregates is also part of a message definition: a synchronization request and response. These add a token and, in some cases, other information.

Open Financial Exchange uses the following naming conventions where the *XXX* message includes:

- Basic request <*XXX***RQ**> and response <*XXX***RS**>
- Transaction wrapper <*XXX***TRNRQ**> and <*XXX***TRNRS**>
- If needed, synchronization wrapper <*XXX***SYNCRQ**> and <*XXX***SYNCRS**>

***NOTE:** Some requests and responses share a transaction wrapper and synchronization wrapper. In these cases, the names of the transaction and synchronization wrappers do not follow the preceding naming conventions.*

2.4.5 Message Sets and Version Control

Message sets are collections of messages. Generally they form all or part of what a user would consider a *service*, something for which they might have signed up, such as "banking." Message sets are the basis of version control, routing, and security. They are also the basis for the required ordering in Open Financial Exchange files.

Within the Open Financial Exchange block, Open Financial Exchange organizes messages by message set. A message set can appear at most once within an Open Financial Exchange block. All messages from a message set must be from the same version of that message set.

2.4.5.1 Message Set Aggregates

For each message set of *XXX* and version *n*, there are two aggregates – one for requests (<*XXX*MSGSRQV*n*>) and one for responses (<*XXX*MSGSRSV*n*>). All of the messages from that message set must be enclosed in the appropriate message set aggregate. In the following example, the Open Financial Exchange block contains a signon request inside the signon message set, and two statement requests and a transfer request inside the bank message set.

```
<OFX>
   <SIGNONMSGSRQV1>                    <!-- Signon message set -->
      <SONRQ>                          <!-- Signon message -->
      ...
      </SONRQ>
   </SIGNONMSGSRQV1>

   <BANKMSGSRQV1>                      <!-- Banking message set -->
      <STMTTRNRQ>                      <!-- Statement request -->
      ...
      </STMTTRNRQ>
```

```
      <STMTTRNRQ>                          <!-- Another stmt request -->
      ...
      </STMTTRNRQ>
      <INTRATRNRQ>                         <!-- Intrabank transfer request -->
      ...
      </INTRATRNRQ>
   </BANKMSGSRQV1>
</OFX>
```

2.4.5.2 Message Set Ordering

Message sets must appear in the following order:

- Signon
- Signup
- Banking
- Credit card statements
- Investment statements
- Interbank funds transfers
- Wire funds transfers
- Payments
- General e-mail
- Investment security list
- FI Profile

The definition of each message set can further prescribe an order of its messages within that message set.

2.4.5.3 Message Set Version Numbers

Message sets have their own version numbers, which are distinct from the version numbers of the Open Financial Exchange headers and the Document Type Definition (DTD) files.

> ***Note:*** *The version numbers of the Open Financial Exchange headers and the Document Type Definition (DTD) files appear in the Open Financial Exchange headers, before the <OFX> data block. For more information about the Open Financial Exchange headers, see section 2.2. The current version number of the headers is OFXHEADER: 100. The current version number of the DTDs is VERSION: 102.*

The following table lists each message set, along with its aggregate name and current version number.

Message Set	*Message Set Aggregate*	*Version Number*
Signon	<SIGNONMSGSETV1>	1
Signup	<SIGNUPMSGSETV1>	1
Banking	<BANKMSGSETV1>	1
Credit Card Statements	<CREDITCARDMSGSETV1>	1
Investment Statements	<INVSTMTMSGSETV1>	1
Interbank Funds Transfers	<INTERXFERMSGSETV1>	1
Wire Funds Transfers	<WIREXFERMSGSETV1>	1
Payments	<BILLPAYMSGSETV1>	1
General e-mail	<EMAILMSGSETV1>	1
Investment security list	<SECLISTMSGSETV1>	1
FI Profile	<PROFMSGSETV1>	1

Note: *For each message set that it is supporting, a financial institution must indicate which version numbers of that message set it supports. The financial institution includes the message set version number in the <MSGSETCORE> aggregate of the FI profile. For more information about the FI profile, refer to Chapter 7.*

2.4.6 Transactions

Other than the signon message, each request is made as a transaction. Transactions contain a client-assigned globally-unique ID, optional client-supplied pass-back data, and the request aggregate. A transaction similarly wraps each response. The response transaction returns the client ID sent in the request, along with a status message, the pass-back data if present, and the response aggregate. This technique allows a client to track responses against requests.

The <STATUS> aggregate, defined in Chapter 3, provides feedback on the processing of the request. If the <SEVERITY> of the status is ERROR, the server provides the transaction response without the nested response aggregate. Otherwise, the response must be complete even though some warning might have occurred.

Clients can send additional information in <CLTCOOKIE> that servers will return in the response. This allows clients that do not maintain state, and thus do not save TRNUIDs, to cause some additional descriptive information to be present in the response. For example, a client might identify a request as relating to a user or a spouse.

In some countries, some transactions require a customer-supplied authorization number. In those countries, the <TAN> element passes this information to servers. As Open Financial Exchange is implemented in each country, the specification will define the requirements for the use of <TAN> in each country.

A typical request is as follows:

Tag	*Description*
<*XXX*TRNRQ>	Transaction-request aggregate
<TRNUID>	Client-assigned globally-unique ID for this transaction, *trnuid*
<CLTCOOKIE>	Data to be echoed in the transaction response, *A-32*
<TAN>	Transaction authorization number; used in some countries with some types of transactions. Country-specific documentation will define messages that require a <TAN>, *A-80*
Request aggregate	Aggregate for the request
</*XXX*TRNRQ>	

A typical response is as follows:

Tag	*Description*
<*XXX*TRNRS>	Transaction-response aggregate
<TRNUID>	Client-assigned globally-unique ID for this transaction, *trnuid*
<STATUS>	Status aggregate
</STATUS>	
<CLTCOOKIE>	Client-provided data, **REQUIRED** if provided in request, *A-32*
Response aggregate	Aggregate for the response
</*XXX*TRNRS>	

List of status code values for the <CODE> element of <STATUS>:

Value	*Meaning*
0	Success (INFO)
2000	General error (ERROR)
2022	Invalid TAN (ERROR)

2.5 The Signon Message Set

The Signon message set includes the signon message, USERPASS change message, and challenge message, which must appear in that order. The <SIGNONMSGSRQV1> and <SIGNONMSGSRSV1> aggregates wrap the message.

2.5.1 Signon <SONRQ> <SONRS>

The signon record identifies and authenticates a user to an FI. It also includes information about the application making the request, because some services might be appropriate only for certain clients. Every Open Financial Exchange request contains exactly one <SONRQ>. Every response must contain exactly one <SONRS> record. Use of Open Financial Exchange presumes that FIs authenticate each customer and then give the customer access to one or more accounts or services. If passwords are specific to individual services or accounts, a separate Open Financial Exchange request must be made for each user ID or password required. This will not necessarily be in a manner visible to the user. Note that some situations, such as joint accounts or business accounts, will have multiple user IDs and multiple passwords that can access the same account.

FIs assign user IDs for the customer. Although the user ID may be the customer's social security number, the client must not make any assumptions about the syntax of the ID, add check-digits, or do similar processing. Servers must accept user IDs, with or without punctuation.

To improve server efficiency in handling a series of Open Financial Exchange request files sent over a short period of time, clients can request that a server return a <USERKEY> in the signon response. If the server provides a user key, clients will send the <USERKEY> instead of the user ID and password in subsequent sessions, until the <USERKEY> expires. This allows servers to authenticate subsequent requests more quickly.

The client returns <SESSCOOKIE> if the server sent one in a previous <SONRS>. Servers can use the value of <SESSCOOKIE> to track client usage but cannot assume that all requests come from a single client, nor can they deny service if they did not expect the returned cookie. Use of a backup file, for example, would lead to an unexpected <SESSCOOKIE> value that nevertheless should not stop a user from connecting.

Servers can request that a consumer change his or her password by returning status code 15000. Servers should keep in mind that only one status code can be returned. If the current signon response status should be 15500 (invalid ID or password), the request to change the password must wait until an otherwise successful signon is achieved.

If the server returns any signon error, it must respond to all other requests in the same <OFX> block with status code 15500. For example, if the server returns status code 15502 to the signon request, it must return status code 15500 to all other requests in the same <OFX> block. The server must return status code 15500 to all requests; it cannot simply ignore the requests.

2.5.1.1 Signon Request <SONRQ>

Unlike other requests, the signon request <SONRQ> does not appear within a transaction wrapper.

Tag	Description
<SONRQ>	Signon-request aggregate
<**DTCLIENT**>	Date and time of the request from the client computer, *datetime*
User identification. Either <USERID> and <USERPASS> or <USERKEY>, but not both.	
<USERID>	User identification string, *A-32*
<USERPASS>	User password on server, *A-171* *NOTE: The effective size of USERPASS is A-32. However, if Type 1 security is used, then the actual field length is A-171.*
-or-	
<USERKEY>	Log in using previously authenticated context, *A-64*
<GENUSERKEY>	Request server to return a USERKEY for future use, *Boolean*
<LANGUAGE>	Requested language for text responses, *language*
<FI>	Financial-Institution-identification aggregate *NOTE: The client will determine out-of-band whether a FI aggregate should be used and if so, the appropriate values for it. If the FI aggregate is to be used, then the client should send it in every request, and the server should return it in every response.*
</FI>	
<SESSCOOKIE>	Session cookie value received in previous <SONRS>, not sent if first login or if none sent by FI, *A-1000*
<APPID>	ID of client application, *A-5*
<APPVER>	Version of client application, (6.00 encoded as 0600), *N-4*
</SONRQ>	

2.5.1.2 Signon Response <SONRS>

Unlike other responses, the signon response <SONRS> does not appear within a transaction wrapper.

NOTE: A client should use DTPROFUP and DTACCTUP only when the service provider that originated SONRS is the same provider that is specified by SPNAME in the profile message set. A

client can determine if the service provider is the same by comparing the value of SPNAME in the appropriate message set with the value for SPNAME in the profile message set.

Tag	*Description*
<SONRS>	Record-response aggregate
<STATUS>	Status aggregate, see list of possible code values
</STATUS>	
<DTSERVER>	Date and time of the server response, *datetime*
<USERKEY>	Use user key instead of USERID and USERPASS for subsequent requests. TSKEYEXPIRE can limit lifetime. *A-64*
<TSKEYEXPIRE>	Date and time that USERKEY expires, *datetime*
<LANGUAGE>	Language used in text responses, *language*
<DTPROFUP>	Date and time of last update to profile information for any service supported by this FI (see Chapter 7), *datetime*
<DTACCTUP>	Date and time of last update to account information (see Chapter 8), *datetime*
<FI>	Financial-Institution-identification aggregate
	NOTE: The client will determine out-of-band whether a FI aggregate should be used and if so, the appropriate values for it. If the FI aggregate is to be used, then the client should send it in every request, and the server should return it in every response.
</FI>	
<SESSCOOKIE>	Session cookie that the client should return on the next <SONRQ>, *A-1000*
</SONRS>	

List of status code values for the <CODE> element of <STATUS>:

Value	*Meaning*
0	Success (INFO)
2000	General error (ERROR)
15000	Must change USERPASS (INFO)
15500	Signon invalid (ERROR); see section 2.5.1
15501	Customer account already in use (ERROR)
15502	USERPASS Lockout (ERROR)

2.5.1.3 Financial Institution ID <FI>

Some service providers support multiple FIs, and assign each FI an ID. The signon allows clients to pass this information along, so that providers know to which FI the user is signing on.

Tag	*Description*
<FI>	FI-record aggregate
<ORG>	Organization defining this FI name space, *A-32*
<FID>	Financial Institution ID (unique within <ORG>), *A-32*
</FI>	

2.5.2 USERPASS Change <PINCHRQ> <PINCHRS>

The signon sends a request to change a customer password as a separate request. The transaction request <PINCHTRNRQ> aggregate contains <PINCHRQ>. Responses are placed inside transaction responses <PINCHTRNRS>. Password changes pose a special problem for error recovery. If the client does not receive a response, it does not know whether the password change was successful or not. Open Financial Exchange recommends that servers accept either the old password or the new password on the connection following the one containing a password change. The password used becomes the new password.

2.5.2.1 <PINCHRQ>

A USERPASS change request changes the customer's password for the specific realm associated with the messages contained in the OFX block. Based on the properties of an OFX profile, defined in Chapter 7, a single OFX block contains instructions related to a single realm. The USERPASS change request thus changes the USERPASS for all message sets associated with one realm. For more information about signon realms, see section 7.2.2.

Tag	*Description*
<PINCHRQ>	USERPASS-change-request aggregate
<USERID>	User identification string. Often a social security number, but if so, does not include any check digits, *A-32*
<NEWUSERPASS>	New user password, *A-171* *NOTE: The effective size of NEWUSERPASS is A-32. However, if Type 1 security is used, then the actual field length is A-171.*
</PINCHRQ>	

2.5.2.2 <PINCHRS>

Tag	*Description*
<PINCHRS>	USERPASS-change-response aggregate
<USERID>	User identification string. Often a social security number, but if so, does not include any check digits, *A-32*
<DTCHANGED>	Date and time the password was changed, *datetime*
</PINCHRS>	

2.5.2.3 <CHALLENGERQ> <CHALLENGERS>

A challenge request is the first step in Type 1 application-level security. Essentially, it asks for some random data from the server. The challenge response provides that server-generated random data and is the second step in Type 1 security.

The challenge message is part of the signon message set and is not subject to data synchronization.

A <CHALLENGERQ> is part of a <CHALLENGETRNRQ> transaction, a <CHALLENGERS> part of a <CHALLENGETRNRS>.

Tag	Description
<**CHALLENGERQ**>	Opening tag for the challenge request.
<USERID>	User identification string, *A-32*
<FICERTID>	Optional server certificate ID. *A-64*
</ **CHALLENGERQ** >	Closing tag for challenge request.

Tag	Description
<**CHALLENGERS**>	Opening tag for the challenge response.
<USERID>	User identification string, *A-32*
<NONCE>	Server-generated random data. *A-16*
<FICERTID>	ID of server certificate used to encrypt. *A-64*
</ **CHALLENGERS** >	Closing tag for challenge response.

When generating the <NONCE>, make sure the data is as unpredictable as possible. See RFC 1750 for recommendations.

The client includes <FICERTID> in the request if it already has the server's certificate. If it's included and matches the server's current certificate, the server may omit the actual certificate from the response.

The server includes <FICERTID> in the response to identify the certificate in a separate MIME part. Even if the certificate itself is not attached, <FICERTID> is still included in the response.

Status code values for the <CODE> element of <STATUS>:

Value	*Meaning*
0	Success (INFO)
2000	General error (ERROR)
15504	Could not provide random data (ERROR)

2.5.2.4 Status Codes

Value	*Meaning*
0	Success (INFO)
2000	General error (ERROR)
15503	Could not change USERPASS (ERROR)

2.5.3 Signon Message Set Profile Information

A server must include the signon message set <SIGNONMSGSET> as part of the <MSGSETLIST> aggregate in the FI profile, since every server must support signon requests.

The information that is part of the <MSGSETCORE> aggregate (for example, the URL and security level) is used only when no other message sets are used. Otherwise, the other message sets override the signon message set for the purposes of batching and routing. For example, if bill payments are sent to a URL that is different from the one used for signon, the client uses the URL specified in the bill payment message set <BILLPAYMSGSET>. For more information about how clients batch and route messages, refer to section 7.1.3.

Tag	*Description*
<SIGNONMSGSET>	Signon-message-set-profile-information aggregate
<SIGNONMSGSETV1>	Opening tag for V1 of the message set profile information
<MSGSETCORE>	Common message set information, defined in the profile chapter
</MSGSETCORE>	
</SIGNONMSGSETV1>	
</SIGNONMSGSET>	

2.5.4 Examples

User requests a password change:

```
<PINCHTRNRQ>
   <TRNUID>888
   <PINCHRQ>
      <USERID>123456789
      <NEWUSERPASS>5321
   </PINCHRQ>
</PINCHTRNRQ>
```

The server responds with:

```
<PINCHTRNRS>
   <TRNUID>888
   <STATUS>
      <CODE>0
      <SEVERITY>INFO
   </STATUS>
   <PINCHRS>
      <USERID>123456789
   </PINCHRS>
</PINCHTRNRS>
```

2.6 External Data Support

Some data, such as binary data, cannot easily be sent within SGML. For these situations, the specification defines a tag that references some external data. The way that clients pick up the external data depends on the transport used. For the HTTP-based transport described in this document, servers can send the data in one of two ways:

- Send the same response, using multi-part MIME types to separate the response into the Open Financial Exchange file and one or more external data files
- Client can make a separate HTTP get against the supplied URL, if it really needs the data

For example, to retrieve a logo, a <GETMIMERS> might answer a <GETMIMERQ> as follows:

```
<GETMIMERS>
   <URL>https://www.fi.com/xxx/yyy/zzz.jpg
</GETMIMERS>
```

If the file includes the same response using multi-part MIME, clients must have the local file, zzz.jpg.

2.7 Extensions to Open Financial Exchange

An organization that provides a customized client and server that communicate by means of Open Financial Exchange might wish to add new requests and responses or even specific elements to existing requests and responses. To ensure that each organization can extend the specification without the risk of conflict, Open Financial Exchange defines a style of tag naming that lets each organization have its own name space.

Organizations can register a specific tag name prefix. (The specific procedure or organization to manage this registration will be detailed at a later time.) If an organization registers "ABC," then they can safely add new tags named <ABC.SOMETHING> without

- Colliding with another party wishing to extend the specification
- Confusing a client or server that does not support the extension

The extensions are not considered proprietary. An organization is free to publish their extensions and encourage client and server implementers to support them.

All tag names that do not contain a period (.) are reserved for use in future versions of the Open Financial Exchange specification.

3. Common Aggregates, Elements, and Data Types

3.1 Common Aggregates

This section describes aggregates used in more than one service of Open Financial Exchange (for example, investments and payments).

3.1.1 Identification of Financial Institutions and Accounts

Open Financial Exchange does not provide a universal space for identifying financial institutions, accounts, or types of accounts. The way to identify an FI and an account at that FI depends on the service. For information about service-specific ID aggregates, see Chapters 11, 12, and 13 on banking, payments, and investments.

3.1.2 Format of User-Supplied Numbers

Clients will not attempt to strip dashes or other punctuation from user-supplied numbers, such as the <TAXID> in an enrollment request or the <*XXX*ACCTTO> in a service-addition request. Servers must be prepared to accept these numbers with or without punctuation.

3.1.3 Balance Records <BAL>

Several responses allow FIs to send an arbitrary set of balance information as part of a response, for example a bank statement download. FIs might want to send information on outstanding balances, payment dates, interest rates, and so forth. Balances can report the date the given balance reflects in <DTASOF>.

Tag	*Description*
<BAL>	Balance-response aggregate
<NAME>	Balance name, *A-32*
<DESC>	Balance description, *A-80*
<BALTYPE>	Balance type. DOLLAR = dollar (value formatted DDDD.cc) PERCENT = percentage (value formatted XXXX.YYYY) NUMBER = number (value formatted as is)
<VALUE>	Balance value. Interpretation depends on <BALTYPE> field, *N-20*
<DTASOF>	Effective date of the given balance, *datetime*
<CURRENCY>	If dollar formatting, can optionally include currency
</CURRENCY>	
</BAL>	

3.1.4 Error Reporting <STATUS>

To provide as much feedback as possible to clients and their users, Open Financial Exchange defines a <STATUS> aggregate. The most important element is the code that identifies the error. Each response defines the codes it uses. Codes 0 through 2999 have common meanings in all Open Financial Exchange transactions. Codes from 3000 and up have meanings specific to each transaction.

The last 10 error codes in each assigned range of 1000 is reserved for server-specific status codes. For example, of the general status codes, 2990-2999 are reserved for status codes defined by the server. Of the banking status codes, codes 10990-10999 are reserved for the server. If a client receives a server-specific status code that it does not know, it will handle it as a general error 2000.

Tag	*Description*
<STATUS>	Error-reporting aggregate.
<CODE>	Error code, *N-6*
<SEVERITY>	Severity of the error: INFO = Informational only WARN = Some problem with the request occurred but a valid response still present ERROR = A problem severe enough that response could not be made
<MESSAGE>	A textual explanation from the FI. Note that clients will generally have messages of their own for each error ID. Use this tag only to provide more details or for the general errors. *A-255*
</STATUS>	

For general errors, the server can respond with one of the following <CODE> values. However, not all codes are possible in a specific context.

Code	*Meaning*
0	Success (INFO)
2000	General error (ERROR)
2021	Unsupported version (ERROR)
15500	Signon error (ERROR); see section 2.5.1

NOTE: *Clients will generally have error messages based on <CODE>. Therefore, do not use <MESSAGE> to replace that text. Use <MESSAGE> only to explain an error not well described by one of the defined codes, or to provide some additional information.*

3.2 Common Elements

This section defines elements used in several services of Open Financial Exchange. The format of the value is either alphanumeric (A-*n*) or numeric (N-*n*) with a maximum length *n*; or as a named type. Section 3.2.8 describes the named types.

3.2.1 Financial Institution Transaction ID <FITID>

Format: *A-255*

An FI assigns an <FITID> to uniquely identify a financial transaction that can appear in an account statement. Its primary purpose is to allow a client to detect duplicate responses. Open Financial Exchange intends <FITID> for use in statement download applications, where every transaction requires a unique ID; not just those that are client-originated or server-originated.

FITIDs must be unique within the scope of the requested transactions (that is, within an account) but need not be sequential or even increasing. Clients should be aware that FITIDs are not unique across FIs. If a client performs the same type of request within the same scope at two different FIs, clients will need to use FI + account + <FITID> as a unique key in a client database.

> ***NOTE:*** *Although the specification allows FITIDs of up to 255 alphanumeric characters, client performance may significantly improve if servers use fewer characters. It is recommended that servers use 32 characters or fewer.*

Usage: Bank statement download, investment statement download

3.2.2 Server-Assigned ID <SRVRTID>

Format: *A-10*

A <SRVRTID> is a server-assigned ID for an object that is stored on the server. It should remain constant throughout the lifetime of the object on the server. The client will consider the SRVRTID as its "receipt" or confirmation and will use this ID in any subsequent requests to change, delete, or inquire about this object.

Where the context allows, a server can use the same *value* for a given server object for both <SRVRTID> and <FITID>, but the client will not know this. SRVRTIDs need to be unique only within the scope of the requests and responses they apply to, such as an account number. Like <FITID>, a <SRVRTID> is not unique across FIs and clients might need to use FI + <SRVRTID> if a client requires a unique key.

Usage: Payments, Banking

3.2.3 Client-Assigned Transaction UID <TRNUID>

Format: *A-36*

Open Financial Exchange uses <TRNUID>s to identify transactions within transaction wrappers (<*XXX*TRNRQ>,</*XXX*TRNRQ>).

In most cases, clients originate <TRNUID>s. When a client originates a <TRNUID>, the value of the <TRNUID> is always set to a unique identifier. Clients expect the server to return the same <TRNUID> in the corresponding response and can use this <TRNUID> to match up requests and responses. Servers can use <TRNUID>s to reject duplicate requests. Because multiple clients might be generating requests to the same server, transaction IDs must be unique across clients. Thus, <TRNUID> must be a globally-unique ID.

In some cases, servers can originate a transaction that was not specifically requested by a client. For instance, a client might set up a recurring payment model. Although the client originates the payment model, the server originates the individual payments. Whenever the server originates a transaction, the value of the <TRNUID> must be set to zero.

The Open Software Foundation Distributed Computing Environment standards specify a 36-character hexadecimal encoding of a 128-bit number and an algorithm to generate it. Clients are free to use their own algorithm, to use smaller TRNUIDs, or to relax the uniqueness requirements. However, it is **RECOMMENDED** that clients allow for the full 36 characters in responses to work better with other clients.

Usage: All services

3.2.4 Token <TOKEN>

Format: *A-10*

Open Financial Exchange uses <TOKEN> as part of data synchronization requests to identify the point in history that the client has already received data, and in responses to identify the server's current end of history. See Chapter 6, "Data Synchronization," for more information.

<TOKEN> is unique within an FI and the scope of the synchronization request. For example, if the synchronization request includes an account ID, the <TOKEN> needs to be unique only within an account. Servers are free to use a <TOKEN> that is unique across the entire FI. Clients must save separate <TOKEN>s for each account, FI, and type of synchronization request.

Usage: All synchronization requests and responses

3.2.5 Transaction Amount <TRNAMT>

Format: *Amount*

Open Financial Exchange uses <TRNAMT> in any request or response that reports the total amount of an individual transaction.

Usage: Bank statement download, investment statement download, payments

3.2.6 Memo <MEMO>

Format: *A-255*

A <MEMO> provides additional information about a transaction.

Usage: Bank statement download, investment statement download, payments, transfers

3.2.7 Date Start and Date End <DTSTART> <DTEND>

Format: *Datetime*

Clients use these tags in requests to indicate the range of response that is desired. Servers use these tags in responses to let clients know what the FI was able to produce.

In requests, the following rules apply:

- If <DTSTART> is absent, the client is requesting all available history (up to the <DTEND>, if specified). Otherwise, it indicates the *inclusive* date and time in history where the client expects servers to start sending information.
- If <DTEND> is absent, the client is requesting all available history (starting from <DTSTART>, if specified). Otherwise, it indicates the *exclusive* date and time in history where the client expects servers to stop sending information.

In responses, the following rules apply:

- <DTSTART> is the date and time where the server began *looking* for information, not necessarily the date of the earliest returned information. If the response <DTSTART> is later than the requested <DTSTART>, clients can infer that the user has not signed on frequently enough to ensure that the client has retrieved all information. If the user has been calling frequently enough, <DTSTART> in the response will match <DTSTART> in the request.
- <DTEND> is the date and time that, if used by the client as the next requested <DTSTART>, it would pick up exactly where the current response left off. It is the *exclusive* date and time in history where the server stopped *looking* for information, based on the request <DTEND> rules.

In all cases, servers are **REQUIRED** to use a "system add datetime" as the basis for deciding which details match the requested date range. For example, if an FI posts a transaction dated Jan 3 to a user's account on Jan 5, and a client connects on Jan 4 and again on Jan 6, the server is **REQUIRED** to return that Jan 3-dated transaction when the client calls on Jan 6.

Usage: Bank statement download, investment statement download

3.2.8 Common Data Types

3.2.8.1 Dates, Times, and Time Zones

There is one format for representing dates, times, and time zones. The complete form is:

YYYYMMDDHHMMSS.XXX [*gmt offset:tz name*]

3.2.8.2 Date and Datetime

Tags specified as type *date or datetime* and generally starting with the letters "DT" accept a fully formatted date-time-timezone string. For example, "19961005132200.124[-5:EST]" represents October 5, 1996, at 1:22 and 124 milliseconds p.m., in Eastern Standard Time. This is the same as 6:22 p.m. Greenwich Mean Time (GMT).

Date and *datetime* also accept values with fields omitted from the right. They assume the following defaults if a field is missing:

Specified date or datetime	*Assumed defaults*
YYYYMMDD	12:00 AM (the start of the day), GMT
YYYYMMDDHHMMSS	GMT
YYYYMMDDHHMMSS.XXX	GMT

Note that times zones are specified by an offset and optionally, a time zone name. The offset defines the time zone.

Take care when specifying an ending date without a time. If the last transaction returned for a bank statement download was Jan 5 1996 10:46 a.m. and if the <DTEND> was given as just Jan 5, the transactions on Jan 5 would be resent. If results are available only daily, then just using dates and not times will work correctly.

> ***NOTE:*** *Open Financial Exchange does not require servers or clients to use the full precision specified. However, they are* ***REQUIRED*** *to accept any of these forms without complaint.*

Some services extend the general notion of a *date* by adding special values, such as "TODAY." These special values are called "smart dates." Specific requests indicate when to use these extra values, and list the tag as having a special data type.

3.2.8.3 Time

Tags specified as type *time* and generally ending with the letters "TM" accept times in the following format:

HHMMSS.XXX[*gmt offset*:*tz name*]

The milliseconds and time zone are still optional, and default to GMT.

3.2.8.4 Time Zone Issues

Several issues arise when a customer and FI are not in the same time zone, or when a customer moves a computer into new time zones. In addition, it is generally unsafe to assume that computer users have correctly set their time or time zone.

Although most transactions are not sensitive to the exact time, they often are sensitive to the date. In some cases, time zone errors lead to actions occurring on a different date than intended by the customer. For this reason, servers should always use a complete local time plus GMT offset in any datetime values in a response. If a customer's request is for 5 p.m. EST, and a server in Europe responds with 1 a.m. MET the next day, a smart client can choose to warn the customer about the date shift.

Clients that maintain local state, especially of long-lived server objects, should be careful how they store datetime values. If a customer initiates a repeating transaction for 5 p.m. EST, then moves to a new time zone, the customer might have intended that the transaction remain 5 p.m. in the new local time, requiring a change request to be sent to the server. If, however, the customer intended it to remain fixed in server time, this would require a change in the local time stored in the client.

3.2.9 Amounts, Prices, and Quantities

3.2.9.1 Basic Format

Format: *A-32*

This section describes the format of numerical values used for amounts, prices, and quantities. In all cases, a numerical value that does not contain a decimal point has an implied decimal point at the end of the value. For example, a numerical value of "550" is equivalent to "550."

The following types are defined to have a maximum of 32 alphanumeric characters, including digits and punctuation. However, clients and servers may have specific limits for the maximum number of digits to the left or right of a decimal point. If a server cannot support a client request due to the size or precision of a number, the server should return status code 2012.

Amount: Amounts that do not represent whole numbers (for example, 540.32), must include a decimal point or comma to indicate the start of the fractional amount. Amounts should not include any punctuation separating thousands, millions, and so forth. The maximum value accepted depends on the client.

Quantity: Use decimal notation.

Unitprice: Use decimal notation. Unless specifically noted, prices should always be positive.

Rate: Use decimal notation, with the rate specified out of 100%. For example, 5.2 is 5.2%.

Some services define special values, such as INFLATION, which you can use instead of a designated value. Open Financial Exchange refers to these as "smart types," and identifies them in the specification.

3.2.9.2 Positive and Negative Signs

Unless otherwise noted in the specification, Open Financial Exchange always signs amounts and quantities from the perspective of the customer. Some typically negative amounts:

- Investment buy amount, investment sell quantity
- Bank statement debit amounts, checks, fees
- Credit card purchases
- Margin balance (unless the FI owes the client money)

Some typically positive amounts:

- Investment sell amount, investment buy quantity
- Bank statement credits
- Credit card payments
- Ledger balance (unless the account is overdrawn)

3.2.10 Language

Language identifies the human-readable language used for such things as status messages and e-mail. *Language* is specified as a three-letter code based on ISO-639.

3.2.11 Other Basic Data Types

Boolean: Y = yes or true, N = no or false.

currsymbol: A three-letter code that identifies the currency used for a request or response. The currency codes are based on ISO-4217. For more information about currencies, refer to section 5.2.

URL: String form of a World Wide Web Uniform Resource Location. It should be fully qualified including protocol, host, and path. A-255.

4. Open Financial Exchange Security

Open Financial Exchange (OFX) provides several options for ensuring the security of customer transactions. This chapter describes the OFX security framework, security goals, types of security, and financial institution (FI) responsibilities.

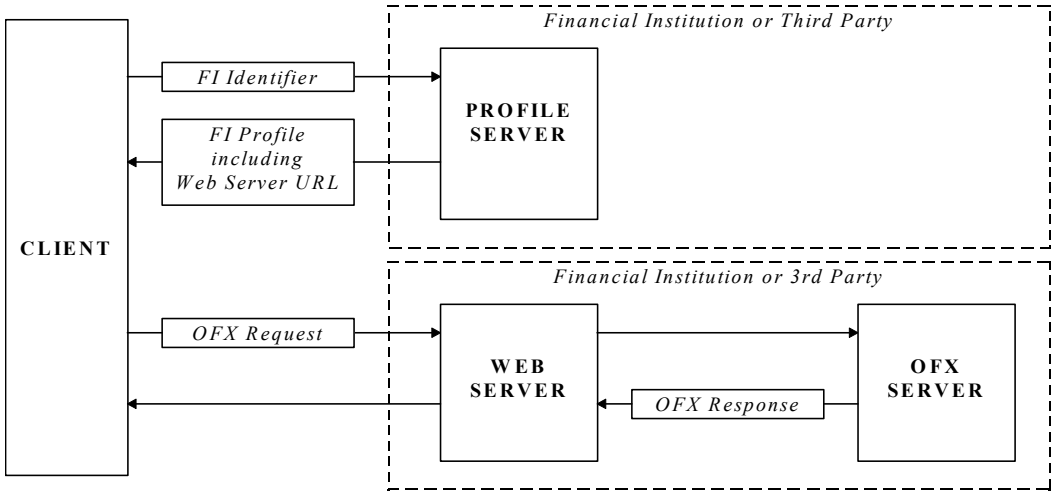
4.1 Security Concepts in OFX

4.1.1 Architecture

Open Financial Exchange security applies to the communication paths between a client and the profile server, a client and the Web server, and — when the OFX server is separate from the Web server, a client and the OFX server. The diagram below illustrates the initial order in which these communications occur, assuming that the client already has the URL for the FI profile server.

The bootstrap process for a client is:

- From the FI Profile Server, the client gets the URL of the FI Web server, so that it can retrieve a particular message set.
- The client sends an OFX request to the FI Web Server URL, from which it is forwarded to the OFX Server.
- The OFX Server sends back a response to the client via the Web Server.



4.1.2 Security Goals

The main goals of Open Financial Exchange security are:

- **Privacy**: Only the intended recipient can read a message. *Encryption* is a technique often used to ensure privacy.
- **Authentication**: The recipient of a message can verify the identity of the sender. In OFX, *passwords* allow an FI to authenticate a client, and *certificates* allow a client to authenticate a server.
- **Integrity**: A message cannot be altered after it is created A cryptographic *hash* is often used to assist integrity verification.

Open Financial Exchange specifies the minimum security required for Internet transactions and provides several security options, based on existing standards. Through its choice of security techniques and related options, an FI can achieve privacy, authentication, and integrity with varying degrees of assurance. For example, there are many kinds of encryption algorithms, most of which can be strengthened or weakened by changing the key size.

4.1.3 Security Standards

Several standards underlie Type 1 security:

- Certificates (X.509 v3) are used to identify and authenticate servers, and to convey their public keys.
- PKCS #1 block type 2 is the encryption format specified by the recipe (See Section 4.2.2.4.3).
- RSA is the encryption algorithm.

4.1.3.1 Certificates and Certification Authorities

A certificate is a digitally signed document that binds a public key to an identity. It contains a public key that identifies information such as the name of the person or organization to whom the key belongs, an expiration date, a unique serial number, and additional descriptive information.

A certificate is useful for authentication because it is signed by a trusted third-party. The assures the verifier that the certificate has not been changed since it was signed. The entity which signs certificates is called a *certification authority*, or CA. A CA acts somewhat like a notary public: the reader of a document stamped by a notary public knows that the notary has checked the identity of the person who originated the document. By digitally signing someone's identity and public key, the CA affirms that the two go together.

If the client and server do not share a common CA, the client cannot validate the server's certificate. For this reason, Open Financial Exchange specifies a number of trusted CAs that all clients must accept and all servers must use.

Certificates are used in Type 1 security, as well as channel-level security through SSL. The format for these is defined by X.509 version 3. For more information, refer to ITU-T Rec. X.509, ISO/IEC 9594-8.

4.1.3.2 PKCS #1

The acronym, PKCS, stands for "Public Key Cryptography Standards," a set of standards developed by a consortium and hosted by RSA. PKCS #1 is the RSA Encryption Standard, the rules for using RSA public key encryption. For the complete syntax of the PKCS #1 standard, refer to "Public-Key Cryptography Standards (PKCS)" published by RSA Data Security, Inc. at http://www.rsa.com/.

4.1.4 FI Responsibilities

Open Financial Exchange is designed with the understanding that there must be a security policy in place at each supporting financial institution. That policy must clearly delineate how customer data is secured, and how transactions are managed such that all parties to the transaction are protected according to accepted and recognized best common practices.

The decision regarding which users may perform a given operation on a given account must be determined by the financial institution. For example, is the specified user authorized to perform a transfer from the specified account? The financial institution must also determine whether the user has exceeded allowed limits on withdrawals, whether the activity on this account is unusual given past history, and other context-sensitive issues.

Although Open Financial Exchange provides many security options, an FI must support a minimal level of security. To ensure the proper security configuration, an FI must follow the steps outlined below.

1. Obtain one certificate for the profile server. This certificate must be rooted in one of the approved Certification Authorities (CAs). Establish appropriate safeguards for this certificate and its private key.
2. Obtain a certificate, rooted in an acceptable CA, for each OFX server, whether it is operated by the FI or by a third party.
3. Decide whether to use Type 1 application-level security for any message sets. For each message set to be secured by Type 1, obtain a certificate.

Type 1 security can be used on any message set, except for the Profile message set.

There are a number of other security issues beyond Open Financial Exchange proper, especially those relating to the Internet and network engineering. These issues are beyond the scope of this document. FIs are advised to conduct a complete security review of all servers associated with Open Financial Exchange.

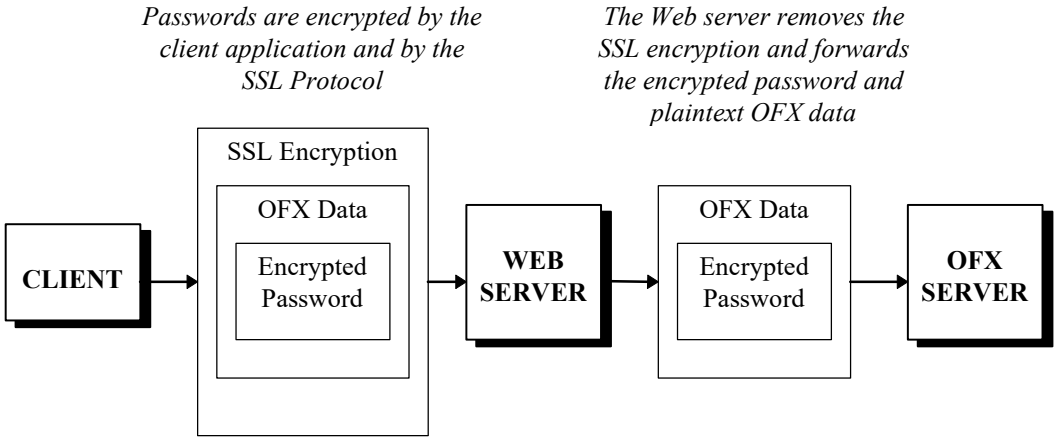
4.1.5 Security Levels: Channel vs. Application

With Open Financial Exchange, security can be applied at two different levels in the message exchange process.

- **Channel level:** Generally transparent to a client or server, channel-level security is built into the communication process, protecting messages between two ends of the "pipe." To secure messages during HTTP transport, client and server applications use the Secure Sockets Layer (SSL) protocol. SSL transparently protects messages exchanged between the client and the destination Web server. SSL authenticates the destination Web server using the Web server's certificate. Additionally, it provides privacy via encryption, and SSL-record integrity, i.e. the block of data sent in each transmission cannot be altered without detection.
- **Application level:** Transparent to and independent of the transport process, application-level security protects the user password sent from the client application all the way to the server application that

handles the Open Financial Exchange messages. The server application typically resides beyond the destination Web server, secured behind an Internet firewall. Application-level security requires channel-level security.

The following diagram illustrates how channel-level and application-level security relate. The diagram shows the path of a request from the client to the server when application-level encryption is used.



Channel-level security is sufficient for most message sets, provided that the network architecture at the destination is adequately secure; however, application-level password encryption can allow a more flexible back-end architecture with a high level of security.

4.2 Security Implementation in OFX

4.2.1 Channel-Level Security

4.2.1.1 Specification in FI Profile

For each message set listed in the FI profile response, the <MSGSETCORE> aggregate describes the channel-level security required for that message set.

The <TRANSPSEC> element defines whether or not channel-level security is required. It can have one of the following values:

Tag	*Description*
N	Do not use any channel-level security
Y	Use channel-level security

All currently defined message sets require channel-level security.

4.2.1.2 SSL Protocol

Secure Sockets Layer (SSL) is a cryptographic protocol commonly used for channel-level security on the Internet. Central to the security of SSL is the *server certificate*. This certificate assures clients that the

server is who it claims to be. It contains the public key of the server, which the client uses to encrypt the session keys it generates as part of each connection.

All of this function is available without significant software development on either the client or server side; however, the client and server must be configured to use appropriate encryption algorithms (CipherSuites). In addition, clients and servers must share a trusted root certificate, or the client will not be able to validate the server's certificate.

> ***NOTE:*** *Although SSL supports client-side certificates to allow a server to authenticate a client, Open Financial Exchange does not require them at this time. To identify and authenticate a customer, servers should use the information provided in the signon request <SONRQ>.*

Setting the <TRANSPSEC> element to Y means that the client must use SSL v3 or higher.

4.2.1.3 Trusted Certificate Authorities

Both channel-level and application-level security rely on clients and servers having at least one trusted certification authority (CA) in common. To ensure that clients can test the validity of a certificate, servers must have their certificates signed by an approved OFX CA[1]. Clients are assumed to have access to this trusted CA.

4.2.1.4 CipherSuites

The following SSL CipherSuites are approved for use with OFX:

- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_IDEA_CBC_SHA
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DH_DSS_WITH_DES_CBC_SHA
- SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_DH_RSA_WITH_DES_CBC_SHA
- SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA

Other CipherSuites are not approved.

---

[1] Approved OFX CAs will be identified at a later date.

4.2.1.5 Key Size

Signing keys must be either RSA with a minimum 1024-bit modulus, or DSS with a 1024-bit modulus.

Server RSA keys and Diffie-Hellman keys must both have a minimum 1024-bit modulus. The Diffie-Hellman base must be primitive.

4.2.2 Application-Level Security

4.2.2.1 Specification in FI Profile

For each message set listed in the FI profile response, the <MSGSETCORE> aggregate describes the security required for that message set.

The <OFXSEC> element defines the type of application-level security required for the message set. <OFXSEC> can have one of the following values, which also are used in the SECURITY element of the OFX headers:

Tag	*Description*
NONE	Do not use any application-level security
TYPE1	Use Type 1 application-level security

Application-level security requires channel-level security.

4.2.2.2 Type 1 Protocol Overview

The goal of the Type 1 protocol is to protect the user password all the way to the destination OFX server. In the absence of client certificates, this password is the primary vehicle for client authentication and is therefore worthy of special consideration.

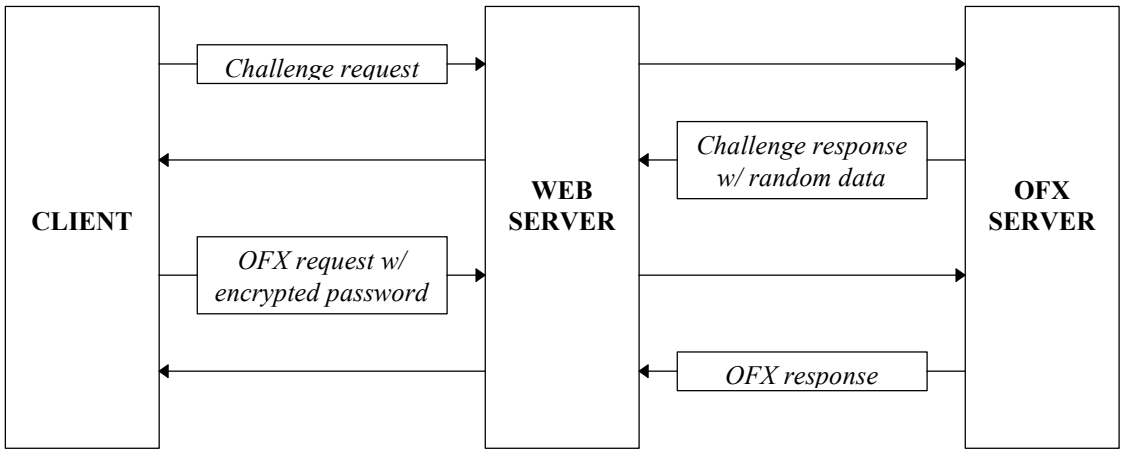
Type 1 requires channel-level security, *i.e.* SSL. Though the password is well protected by SSL alone in the client to Web server connection, the server-side network architecture may render the password less secure while it is in transit between the Web and OFX servers. With Type 1, the user password is not decrypted until the request reaches the OFX server.

Type 1 applies only to the request part of a message; the server response is unaffected.

A simple approach would be to deliver the server's Type 1 certificate in the profile and use it to encrypt the password, but that would permit a *replay attack*. An attacker could capture a transaction, including encrypted password, and replay it to the server. It wouldn't matter that the password remained unknown.

To prevent the *replay attack*, the server introduces some random data to the process, data which is unpredictably different for each transmission. The client asks for the random data with a challenge request. The server sends it, along with its Type 1 certificate, in the challenge response. The client then uses that random data in the encryption process, thereby assuring the server that the client response is associated with this and only this interaction.

The following diagram illustrates:



4.2.2.3 Type 1 Protocol Notation

In this section, the expression , $C = E_A (M)$, means that plain text M is encrypted either symmetrically or asymmetrically with key A into ciphertext C. The expression, $M = D_A(C)$ signifies the inverse operation (decryption), in which ciphertext C is decrypted into plain text M using key A. If C was encrypted asymmetrically, then A in the latter case is understood to be the private component of the key. The expression, A || B, indicates that B is concatenated to A.

4.2.2.4 Type 1 Protocol Implementation

Type 1 application-level security provides additional password secrecy. These are the steps for conducting a Type 1 transaction (unless otherwise noted, the term "Server" in this section refers to the Financial Institution Server):

1. Client obtains the Server's profile from the Profile Server (see Profile chapter)
2. Client establishes an SSL connection with the Server (see Section 4.2.1)
3. Client sends <CHALLENGERQ> to Server (see Section 4.2.2.4.1)
4. Server sends <CHALLENGERS> which contains a nonce and the Server's Type 1 certificate (see Section 4.2.2.4.2)
5. Client builds a transaction request and sends it to the Server (see Section 4.2.2.4.3)
6. Server parses the request, verifying the user password, and either rejects or processes the transaction (see Section 4.2.2.4.4)

The following table lists data elements used in the Type 1 protocol:

Field	Type	Description
BT	octet, length 1	Block Type byte. BT = 0x02
CT1	octet string, length 128	Ciphertext: the PKCS #1 RSA encryption of EB with KS. CT1 = E_{KS}(EB)
CT2	printable ASCII, length 171	Encoded Ciphertext: the RADIX-64 encoding of CT1 (see RFC 1113, §4.3.2.4 and §4.3.2.5). CT2 = RADIX64(CT1)
D	octet string, length 68	Data: the user data to be encrypted. D = NC || P || T
EB	octet string, length 128	Encryption Block: the formatted plain text block, ready for encryption. EB = 0x00 || BT || PS || 0x00 || D
KS	RSA key, modulus length 1,024 bits	Server's Type 1 RSA key
NC	octet string, length 16	Client Nonce: string of random octets generated by the Client
NS	octet string, length 16	Server Nonce: string of random octets generated by the Server
P	printable ASCII, null-padded, length 32	Password: shared by the Client and Financial Institution, null-padded on the right
PS	octet string, length 57	Padding String: each octet is pseudo-random and non-zero
T	octet string, length 20	Authentication Token. T = SHA1(NS || P || NC)

```
struct {
      unsigned char nc[16];
      unsigned char p[32];
      unsigned char t[20];
} D;
struct {
      unsigned char null1 = 0x00;
      unsigned char bt = 0x02;
      unsigned char ps[57];
      unsigned char null2 = 0x00;
      struct D d;
} EB;
```

4.2.2.4.1 Challenge request

Client sends a <CHALLENGERQ> to the Server.

4.2.2.4.2 Challenge response

Server sends a <CHALLENGERS> to the client. This response contains the Server's Type 1 certificate and NS.

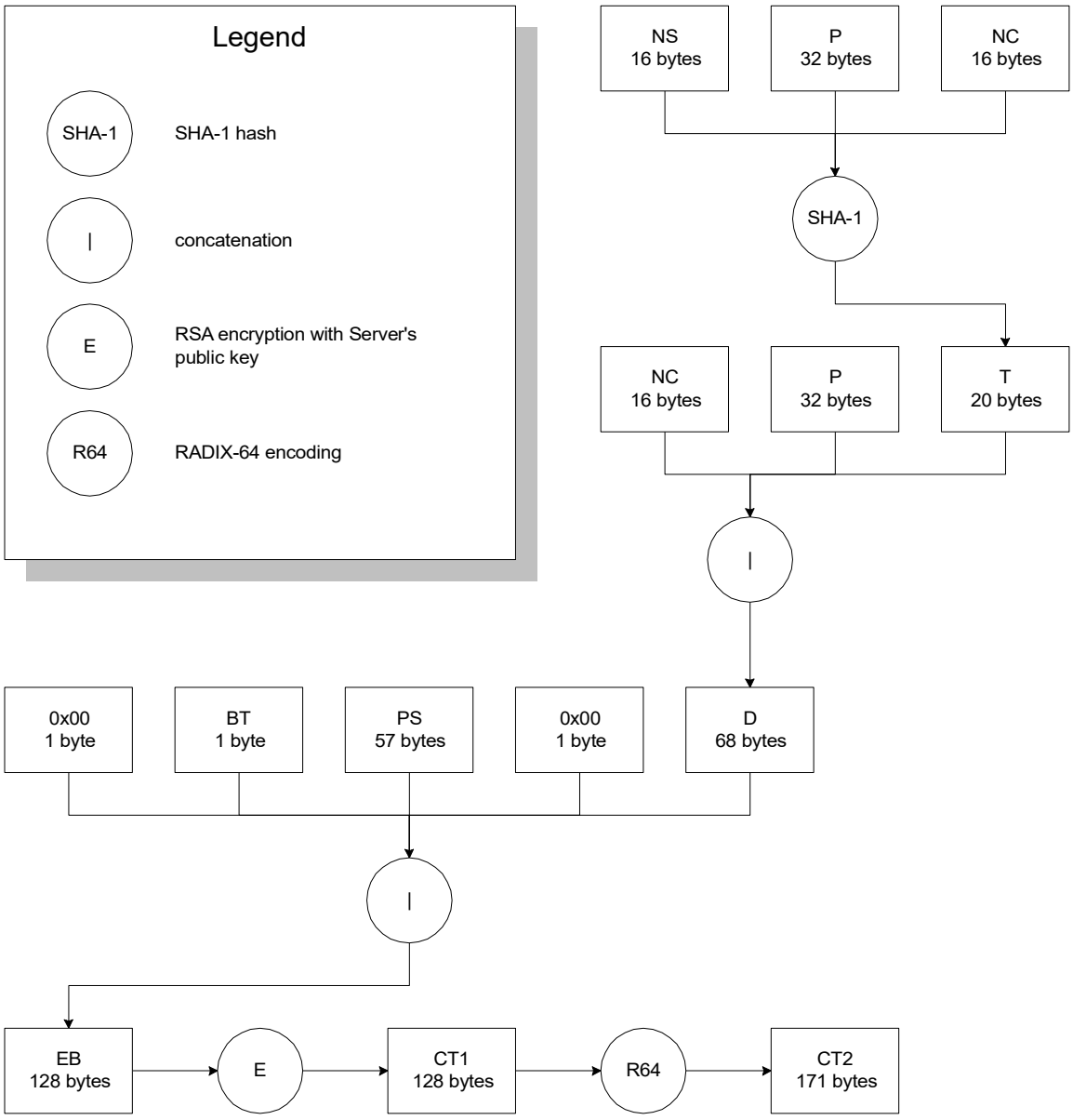
4.2.2.4.3 Building the OFX Request

1. Client generates 16 random octets and places them in NC (see RFC 1750 for recommendations on entropy generation)
2. Client obtains the User's password (P)
3. Client computes T = SHA1(NS || P || NC)
4. Client generates 57 pseudo-random, non-zero octets and places them in PS (NC may be used to seed the pseudo-random number generator)
5. Client sets D = NC || P || T
6. Client sets EB = 0x00 || BT || PS || 0x00 || D
7. Client RSA-encrypts EB using the Server's Type 1 public key (obtained from the Server's Type 1 certificate): CT1 = E_{KS}(EB) (see PKCS #1, §§8.2-8.4)
8. Client encodes the ciphertext for transport: CT2 = RADIX64(CT1). See RFC 1113, §4.3.2.4 and §4.3.2.5. This is a standard encoding method supported by RSA's Bsafe library and others.
9. Client constructs the body of its OFX request
10. Client copies CT2 to the <USERPASS> field of the OFX <SONRQ>

11. Client sends the complete OFX request to the Server

In <PINCHRQ>, the steps are identical, except that in step 2, P is set to <NEWUSERPASS> and in step 10, CT2 is copied to the <NEWUSERPASS> field of the <PINCHRQ>.

The diagram below illustrates the creation of CT2.



4.2.2.4.4 Parsing the OFX Request

1. Server reads the <SECURITY> field in the OFX header to ascertain whether Type 1 processing should be used on this message. If Type 1 is not used, skip to step 7
2. Server extracts CT2 from the <USERPASS> field of the OFX <SONRQ> and removes the encoding to obtain CT1 (see RFC 1113, §4.3.2.4 and §4.3.2.5)
3. Server decrypts CT1 to obtain EB: EB = D_{KS}(CT1) (see PKCS #1, §9)
4. Server extracts D from EB, then extracts NC, P, and T from D
5. Server looks up the Client's password in its database, and computes SHA1(NS || P || NC). If the result does not match T, Server terminates the session and reports the error to the client
6. Server processes the request and returns confirmation to the Client

In <PINCHRQ>, the steps are identical except that in step 2, CT2 is obtained from the <NEWUSERPASS> field of the OFX <PINCHRQ>.and in step 5, the server does not look up the extracted new password in a database.