1. Introduction

The Simple Commerce Messaging Protocol (SCMP) is a general-purpose commerce transport protocol for securely communicating a set of data from a sending agent’s application to a receiving agent’s server. And, where the response by the receiving agent’s sever to the sending agent is, in fact, the reply from the request represented by the set of data in the message’s payload. The intent of this protocol is to define a method where trading partners can perform on-line business requests in an environment where the sending partner is fully authenticated, and the message cannot be repudiated.

The SCMP message content, hereinafter referred to as message payload, is not intended to be defined or specified. SCMP does not specify payload contents or how trading partners are expected to process the payload, beyond basic server-level functions related to processing SCMP-headers. This intent is to permit trading partners the flexibility to implement either a standard commerce message format as in ANSI-X12 Electronic Data Interchange (EDI) or some other proprietary request format.

The only requirement on the message payload is that it be identified to the receiver utilizing MIME naming and formatting [MIME] and used to identify the type of payload contained in the SCMP data area.

In this manner, SCMP fundamentally differs from many emerging commerce message protocols. Beyond specifying the method for transport,
encryption, authentication and handling, these other protocols specify the contents of the message and details how a server is to process and respond to the message payload.

SCMP is intended as both an on-line and batch protocol. The exact content type of the message and the processing constraints are specified in SCMP-headers.

1.1. Document Overview

This document describes SCMP from the standpoint of how trading partners would implement a client/server request processing system where a sending agent requests services via an untrusted network connection from a server-based receiving agent.

In this environment, the typical requirements for authentication, non-repudiation, message integrity, and privacy as discussed in [SMIME] and assured by the proper use of the Secure/Multipurpose Internet Mail Extensions [SMIME]. Beyond this, the trading partners require service-based extensions to standard MIME and SMIME security services. These service-based extensions are described within this document, while it is assumed the trading partner will implement MIME and SMIME services as described in [MIME] and [SMIME] respectively.

1.2. Terminology

Throughout this draft, the terms MUST, MUST NOT, SHOULD, and SHOULD NOT are used in conformances to the definitions in [MUSTSHOULD].

1.3. Definitions

Several terms will be used when specifying SCMP.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trading Partners</td>
<td>Two entities wishing to perform some on-line request processing where authentication, privacy, integrity and non-repudiation of the requests are important. Tranding partners have established a trusted relationship between each other.</td>
</tr>
<tr>
<td>Client</td>
<td>An application program that executes on a remote system, used by a trading partner to request services from a server via an un-trusted or publicly switched packet network, like the Internet.</td>
</tr>
<tr>
<td>Server</td>
<td>An application program used to process SCMP messages received from a client, and generate appropriate replies which are sent back to the client.</td>
</tr>
<tr>
<td>Request</td>
<td>A discrete unit of service embodied in a single SCMP message/reply pair.</td>
</tr>
<tr>
<td>Payload</td>
<td>The meaningful content provided by a client to a server, encapsulated in an SCMP message. Similarly the meaningful content provided by a server to a client, encapsulated in an SCMP message.</td>
</tr>
<tr>
<td>Request</td>
<td>An SCMP message sent from a client to a server.</td>
</tr>
<tr>
<td>Reply</td>
<td>An SCMP message sent from a server to a client.</td>
</tr>
<tr>
<td>Services</td>
<td>Algorithms implemented by the server application which are executed as designated by the payload.</td>
</tr>
</tbody>
</table>
Each available algorithm is a service.

2. Payload Encapsulation

The payload of an SCMP message MUST be prepared as a standard MIME entity as defined in the [MIME] specification. The [SMIME] document describes how the resulting MIME entity SHOULD be cryptographically enhanced according to [CMS], which is derived from PKCS #7, [PKCS-7].

An SCMP compliant server SHOULD implement the three message types as described in [SMIME], signed, enveloped, and signed/enveloped. An SCMP compliant server MUST implement signed/envelope message type as described in [SMIME].

It is recommended, for non-repudiation concerns that the trading partners SHOULD exchange signed or signed/enveloped SCMP message types.

It is also recommended that strong enough cryptographic methods be used to insure authenticity, integrity, non-repudiation, and privacy of the payload. But, if the trading partners form a private agreement, clear data or signed-only data MAY be exchanged. However, an SCMP compliant server MUST support encryption even if encryption is not being used.

In addition to the standard MIME headers, a compliant implementation MUST define "SCMP-protocol-version" and "SCMP-protocol-sender-name". These headers added to the outer MIME entity, as described in [SMIME].

2.1 SCMP Protocol Version:

The SCMP-protocol-version header is used to designate the SCMP protocol version. Therby the protocol version can be accessed before any decryption of the request is performed.

An example SCMP-protocol-version header will be in this format:

    SCMP-protocol-version: v2.0

The possible protocol versions MUST be agreed upon by the trading partners.

2.2 SCMP Sender Name:

The SCMP-sender-name header is used to designate the SCMP sender name. Therby the sender name can be accessed before any decryption of the request is performed.

An SCMP-sender-name header will be in this format:

    SCMP-sender-name: CyberSource

The possible sender names MUST be agreed upon by the trading partners.

Use of the remaining standard SMIME (outside MIME entity) headers are assumed. This includes any additional implementation-specific headers. These headers will most likely be ones that need to be processed prior to payload decryption.

3. SCMP Payload-based Headers

This section describes the service-based extensions that MUST be implemented by both the client and server to insure correct and proper request processing. Processing the SCMP service headers is the responsibility of the application processing the request. The following headers are described for the payload of the S/MIME entity, and MUST be prepared as defined in [MIME]. Thus if the S/MIME message type is signed/enveloped (which is recommended), then the SCMP headers will be encrypted to protect the privacy of the sender.
3.1. Request Time to Live:

This describes the amount of actual processing time in seconds the client expects the server to fully complete payload processing prior to responding with an appropriate reply.

An SCMP server receiving a SCMP message MUST evaluate the request time to live value and determine if it can execute the required service(s) in the amount of time designated. Assuming the server believes it can complete the work within the allowed time, it will accept the request. If not, it MUST return an error to the client stating it could not accept the request.

Once a server has accepted a request, it MUST process it until the time to live value has been reached or until completion. If the time to live value is reached during execution, the server MUST return an error to the client stating that a timeout has occurred. Measures to ensure data integrity after the time to live value has been exceeded will be the responsibility of the implementation.

The time to live header will be in this format:

```
SCMP-request-time-to-live: 0..n (seconds)
```

A client, having received a time-out error message, SHOULD send a "request status message" to the server, referencing the original scmp-request-id (from the message that timed out) in the message payload. The server’s reply to this status message would be the reply that would have been sent had the processing time not exceeded the time to live metric.

3.2 Request Type:

This header describes the processing mode of the request. Two modes are supported "batch" and "real". Wherby "batch" mode does not imply any limit on request processing time. "real" mode requires a limit in processing time as specified by scmp-request-time-to-live header.

The request type header will be in this format:

```
SCMP-request-type: batch|real
```

An SCMP-request-type value of "batch" MUST cause the server to respond with an acknowledgement reply to the client. The server SHOULD then process the message according to an appropriate schedule and respond to the client as appropriate after completing the actual processing.

3.3 Request Return Path:

This describes the method used to respond to the client after completion of a "batch" request. The client will not be notified upon "batch" request completion if the return path is empty. The return path value MUST conform to the Uniform Resource Locator specification. [URL]

An example return path would be in this format:

```
SCMP-request-return-path: http://www.batch.batch.com:8080
```

The list of protocols supported MUST be furnished via a private agreement between trading partners.

3.4. Message Type:

This value specifies the type of payload that is contained in the SCMP
message. The intent of this header is to provide a meta-level description of the message payload and allow a receiving server to decide which services or associated algorithms to use in processing the payload. The message type value SHOULD NOT be the sole specifier of the services being requested by a client from a server.

Message type is specified as follows:

```
SCMP-message-type: [service-name]/[version-number]
```

The assignment of service names MUST be provided by the server to a client at the time a service is published. For instance, if a service was published called "CommerceService", the SCMP-message-type might be represented as:

```
SCMP-message-type: CommerceService/1.0
```

It is assumed that trading partners will agree on service names before request are processed. Additionally servers MUST allow service names to be configurable, regardless of what the algorithm which implements the service does.

3.5. Request ID:

A value in the format described in [822] for the Message-ID header with the left part constrained to be a 22 character string value. Request ID’s MUST be generated by the client application.

An example of a request scmp-request-id is:

```
scmp-request-id: 0917293049096167904518
```

The scmp-request-id MUST be unique and SHOULD NOT be easy to predict to prevent a potential denial of service attack. A client application when preparing the scmp-request-id, SHOULD perform a random number generation with sufficient degrees of randomness so as to ensure uniqueness and unpredictability of the result.

Servers MAY use a scmp-request-id as a reference and handle to the original request.

4. SCMP Data Block (Message Payload)

The payload or data block can be any arbitrary data type in the format as specified by the SCMP-message-type. This payload forms the content of the SMIME message as described in [SMIME].

5. Certificates

Every trading partner implementing SCMP MUST exchange certificates that have been issued and signed by one or more mutually trusted certificate authorities (CA). These certificates are used to guarantee the authenticity of public keys. Prior to establishing trading relationships on the basis of SCMP request, sender and receiver MUST have acquired mutually acceptable trusted public root certificates in a trusted, secure, out-of-band manner.

Trading partners, upon receiving or exchanging public key certificates for the first time, SHOULD validate the certificate and certificate chain before processing an SCMP request.

It is also recommended that the trading partners re-validate any certificates and certificate chains on a scheduled basis.
Upon establishing a relationship between trading partners, the recipient of a new certificate (the server in most cases) SHOULD validate the certificate as soon as is practically possible. Certificate re-validation policy, related to the frequency known certificates are revalidated against a certificate authority’s certificate revocation list, is not specified by SCMP. This matter is left as a policy decision for the operator of the SCMP server.

6. Transport Implementations

SCMP can be implemented using any variety of transport methods as agreed between trading partners. Here are a few examples.

http: This delivers a SCMP message to a server URL and should use a POST function.

electronic mail: This will support a queued batch processing service

7. Receiving Server Functions

This section describes minimal server functions required to implement SCMP.

7.1. General

A SCMP server receives a message from a client, processes the message and generates a reply. If the message type is signed or signed/enveloped the server initially validates the outer signature. If the outer signature is not valid the server MUST NOT process the request further.

7.1.1. Message Timestamp

The time a request was sent will be derived from the standard SMIME date header. If a client is specifying a time to live the client SHOULD be synchronized using [NTP] or Secure NTP. The sender of an SCMP message will place the time a message was dispatched into the SMIME header in [MIME] format. The message timestamp SHOULD be included in the SCMP payload if possible and used, in combination with the scmp-request-id, by the server to prevent a replay attack.

It is recommended that servers run a client-visible NTP server to allow sending agents running SCMP client applications to synchronize clocks as required.

7.2. Application issues

The server SHOULD evaluate the signature of the message, (if the message is of signed or signed/enveloped type), prior to processing the message payload. Within this process the server SHOULD obtain the senders certificate via. the distinguished name in the certificate as described in [HOUSLEY].

Assuming the SCMP message’s signature is valid, the server will process requests based on the SCMP-message-type value.

7.2.1. Request Serialization

A server may not guarantee serialized request processing. If requests must be serialized, it is expected that all of the serialized transactions will be received in a single message payload or that other content specific serialization systems will be used.
7.2.2. Server Errors

A server may encounter several classes of error conditions. The server MUST be capable of reporting an error as described in section 8 of this document. Detection may vary based on specific implementation.

A server MUST be capable of detecting a duplicate scmp-request-id and notify the sending client application of the duplicate request. Duplicate request detection MUST be based on the scmp-request-id and the distinguished name of the signer to prevent denial of service attacks. Servers MUST take steps to prevent error conditions in which request retries overlap the original request processing. In this case the server MUST NOT respond to the retry until the original result is available.

In the event of a duplicate request being detected the server MUST:
1) lookup the prior request
2) verify the sender is the same
3) return an appropriate error message to the client.

In the event that the three above steps fail, the server MUST return an appropriate SCMP error message.

8. Protocol Level Error Messages

In general SCMP does not concern itself with application level errors. Such errors MUST be returned in an SCMP reply with appropriate application specific formatting.

8.1. Format

SCMP error messages are returned by a server as signed data. SCMP errors MUST NOT be encrypted to permit clients to process encryption related errors.

The format of SCMP errors is:

    SCMP <error number> <error message text>

8.2. Client Application Error Handling

Client action in the case of error return is error specific and not defined. If the server fails to return any reply within twice the time to live requested (due to unspecified server or network failure) the client SHOULD re-send the request. Upon receipt of a duplicate request the server will respond as described in 7.1.3. Clients MUST NOT retry a request in less than the time to live interval of the original request.

9. Author’s Address

Tom Arnold
CyberSource Corporation
550 S. Winchester Blvd., #301
San Jose, CA 95128
E-mail: dptom@cybersource.com
Phone: 408-556-9100

Jason Eaton
CyberSource Corporation
550 S. Winchester Blvd., #301
San Jose, CA 95128
E-mail: jeaton@cybersource.com
Phone: 408-556-9100
Michael Jimenez  
CyberSource Corporation  
550 S. Winchester Blvd., #301  
San Jose, CA 95128  
E-mail: mjimenez@cybersource.com  
Phone: 408-556-9100

Hubert Chen  
CyberSource Corporation  
550 S. Winchester Blvd., #301  
San Jose, CA 95128  
E-mail: mjimenez@cybersource.com  
Phone: 408-556-9100

10. Acknowledgements

Several persons (in alphabetic order) have contributed, and continue to contribute significantly through their participation in an industry-based EMAP (Electronic products Messaging and Protocol) committee. Through their participation and continued support SCMP will continue to develop into a functioning, on-line Internet commerce messaging protocol. Their contributions are greatly appreciated.

Ron Bose (LitleNet), Len Cantor (IBM), Hubert Chen (CyberSource), Tony Curwen (Ingram Micro), Mike Myers (Verisign), John Pettitt (Beyond.com), Jesse Rendleman (CyberSource), Don Sloan (Tech Data), Frank Tyksen (Portland Software), Paul Guthrie (VISA)

11. References


[MUSTSHOULD] "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, IETF.

