NCHELP

Technical Manual

Original Issue: 03/30/2004
Revised: 01/31/2007
Revised: 10/01/2008

Effective: 10/01/2008

TABLE OF CONTENTS

INTRODUCTION 1
Electronic Exchange Compliancy Regulation Standards 1
Overview 1
Revisions and updates 2

Security and Authentication: Mechanics 3
Introduction 3
Encryption 4
SSL Encryption 5
OpenPGP Encryption 5
Digital Signatures 6
Diagrams 8
Diagram 1: PGP Encryption & Signing 8
Diagram 2: PGP Decryption & Signature Verification 8
Diagram 3: DTS v1.0.1 - Digital Signature 9
Diagram 4: DTS v1.0.1 – Verifying Digital Signature using X.509 Certificate 10

Security and Authentication: Key Management 12
Key Generation 12
OpenPGP 12
X.509 Certificates 15
Key exchange 16
OpenPGP 16
X.509 Certificate 17
Verifying public key/certificate exchanges 18
OpenPGP 18
X.509 Certificate 18
Certificate Authority signing process 19
Verification out of band via HTTPS (optional) 19
Database of Public Keys / Key Ring 22
Updating keys 22
Deleting keys 23
Diagrams 24
Diagram 1: Exchanging OpenPGP format Public Keys 24
Diagram 2: Exchanging X.509 certificates 25
Diagram 3: Obtaining Certificate Authority signed X.509 Certificate 26

TRANSPORT PROTOCOLS: SMTP/POP3 27
Introduction 27
Conceptual overview 27
SMTP/POP3 business partner responsibilities 28
SMTP/POP3 SBS vendor responsibilities 29
E-mail system requirements 30

E-mail subject line components....30
Key identifier (X.Y.Z) 31
Unique message identifier (M)....31
E-mail body....33
File attachment 33
File attachment size 33
Naming a file attachment....33
E-mail message acknowledgment (Evaluate for removal)....34
Message acknowledgement timeout....35

TRANSPORT PROTOCOLS: File Transfer (FTP) 37
Introduction....37
FTP business partner responsibilities....39
FTP server requirements 41
FTP security....41
FTP directory structure....41
FTP filename components 43
File types (A)....44
Encryption protocol code (B) 44
Key identifier (X_Y_Z) 44
Unique message identifier (M)....44
FTP file sizes 46
Archiving FTP files....46
Acknowledging FTP files....46
Purging archived FTP files 47

TRANSPORT PROTOCOLS: Data Transport Standard 48
Introduction....48
DTS Web Server Requirements....52
DTS Security and Authentication 52
Required Encryption Keys 52
Supporting Digital Signatures 52
DTS Payload sizes....53
DTS End-Points 53
DTS Business Partner Responsibilities 53
DTS Client....54
DTS Service 54
Timeout methodology for DTS Client and Service implementation....54
DTS SOAP Components....55
DTS Header elements 55
DTS Body Element....58
DTS Error Handling 58
DTS Transaction Flows....59
DTS Client....59
DTS Service 61

Valid Values for Components and Examples 65
Valid Values....65

Encryption Protocol Code 65
File Type Component and Payload Type Header 65
Examples 70
Email Subject line 71
FTP filename examples 72
DTS SOAP Examples 75

APPENDIX A: OPENPGP SPECIFICS 79
GNU Privacy Guard specifics 79
Options file 79
GNU Privacy Guard commands 80
PGP E-Business Server specifics 85
Options file 85
PGP commands 86

APPENDIX B: PAYLOAD ROUTING 88

APPENDIX C: SUPPORTING DOCUMENTATION 92
RFCs 92
RFC Web sites 92
RFC references 92
Federal Information Protection Standards (FIPS) 93
FIPS Web sites 93
GNU General Public License (GPL) 93
GPL Web Sites 93
GNU Privacy Guard 93
CommonLine and CAM information 93

APPENDIX D: Considerations for “Push-Pull” implementation 94

Appendix E: Use of Response “Tokens” in DTS 96

GLOSSARY 98

THIS PAGE IS INTENTIONALLY BLANK

INTRODUCTION

Electronic Exchange Compliancy Regulation Standards

This document defines constraints and limitations that are recognized as standard requirements by the Electronic Exchange community. Exceeding these standardized constraints and limitations places an organization at odds with the Electronic Exchange community's various standard-based implementations. No entity is permitted to require another entity to exceed these standardized constraints and limitations. Should any two entities agree to exceed these standardized constraints and limitations, they are no longer considered to be operating under the Electronic Exchange standards and are then operating under what is considered to be a proprietary implementation. While these arrangements are not prohibited, the details of these arrangements can not be forced on non-agreeing entities. In short, if an entity chooses to exceed these limitations, the entity must remain capable of interacting with standards compliant implementations.

Overview

The NCHELP Technical Manual is a tool for communicating technical information to organizations participating in the CommonLine, Common Account Maintenance (CAM) and Common Record: CommonLine (CR:C) processes. **This document is not tied to a specific release of any underlying file format. It can and will be republished as technical specifications change and new technical needs arise. This version of the Technical Manual (revised 10/01/2008) replaces the updated version (issued 1/31/2007) and the previous version published on 03/30/2004.**

The changes indicated in this version of the Technical Manual have been reviewed and approved by the Electronics Standards Committee (ESC). Unless noted otherwise, all changes specified in this version of the Technical Manual are effective immediately.

This document discusses the following topics:

- CommonLine, CAM, CR:C, and other optional file transport protocols

  The file transfer sections describe the transfer protocols available for use with the CommonLine, CAM and CR:C processes. Certain transfer protocols are required based on the process used (CommonLine, CAM or CR:C); however, optional transfer protocols may be used based on agreements between trading partners.

- Encryption

  The encryption section addresses data exchange security. While the encryption methods is not necessarily specific to a particular data exchange protocol, some aspects of encryption as described are specifically related to the transport protocols discussed elsewhere in this document.

Technologies discussed in this manual are explained by the reference sources listed in ***Appendix C Supporting documentation.***

Revisions and updates

1/31/2007 – General significant revision and reformatting of the original document. Topics that were contained within the previous document are represented within this revision; however they may be represented in new or different ways.

10/1/2008 – Revision of encryption standards and recommendations for secure FTP.

Security and Authentication: Mechanics

Introduction

CommonLine, CAM and CR:C participants must be assured that transmissions sent and received over the Internet will remain secure. Adequate security is achieved through the use of encryption and digital signatures. Encryption restricts unintended recipients from viewing the data; the digital signatures prohibit altering the message or impersonating the sending organization. There are two forms of Encryption explained in this section: OpenPGP, utilizing asymmetric encryption keys that conform to OpenPGP (RFC 2440) standards and Secure Sockets Layer (SSL), utilizing X.509 certificates under the SSL specification.

There must be only one set of OpenPGP asymmetric keys for each encryption protocol code, per organization ID (as defined by ED or NCHELP) and non-ED branch ID. If an organization uses non-ED branch IDs, there may be separate keys for each organization/non-ED branch ID combination if desired. See OpenPGP/Key Generation section for details regarding how the Organization ID and non-ED Branch ID are related to the keys. The OpenPGP compliant keys are used for both SMTP/POP3 and FTP file transmissions. The X.509 certificates are used for the Data Transport Standard.

NOTE

In the case of X.509 certificates, there is a specific situation where two certificates per organization may be used. In this case, one certificate would be utilized for SSL and one for generating the Digital Signature. This situation is discussed in the X.509 subsection of the Security and Authentication: Key Management chapter.

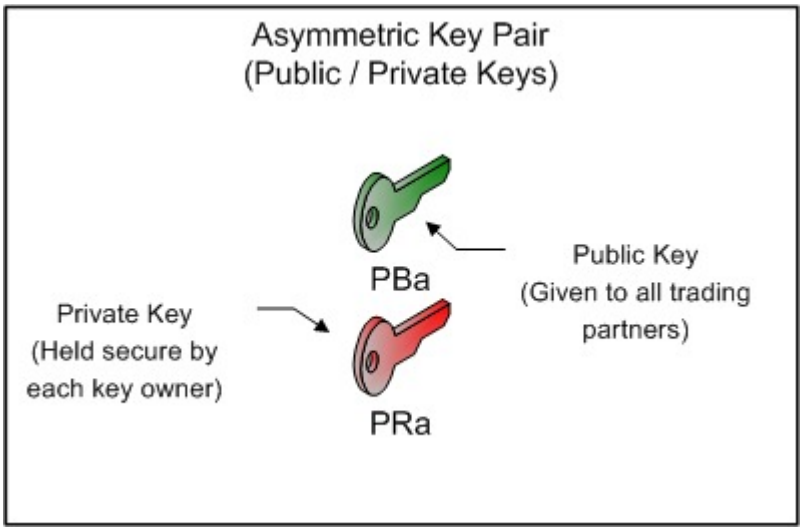
Encryption

Encryption schemes fall into two categories: symmetric key and asymmetric key. The symmetric key method uses the same key to encrypt and decrypt a message. This method requires the sender and receiver of the file to share the same secret key. There are numerous security and logistical problems with this method of encryption.

The asymmetric key (or "public key") method uses one key to encrypt the message and uses another key to decrypt the message. In this two-key method, each organization has one public key and one private key. The encryption key (or public key) is provided to everyone in a trading network. The decryption key (or private key) is only held by the receiving organization. The key that is exchanged, the public key, can encrypt a message but is not sufficient to decrypt it. Thus, the public key can be made freely available. Its complement, the private key, is held secure by the receiving organization and is used to decrypt the message.

The two key pairs are also used for creating and encrypting the digital signature, but in reverse roles from that used for encryption. The sender uses their private key for signing and encrypting the signature. The recipient uses the sender's public key for verifying that signature. (See Digital Signature section below)

The following diagram illustrates public and private keys:



One of the attractions of the public key method of encryption is the limitation of damage in the case of a key compromise. If a private key is compromised, the only part of the CommonLine, CAM, or CR:C trading network affected is that of the key's owner and their trading partners. To restore security, the compromised participant must generate a new public and private key pair and distribute their new public key to all trading partners (See Updating and Deleting sections). While this situation requires new keys to be generated for the participant and distributed to their trading partners, the overall network is not compromised.

The asymmetric key method of encryption utilizing OpenPGP has been selected for CommonLine, CAM and CR:C file transfers for the POP/SMTP and FTP protocols. OpenPGP is

a file-oriented encryption method, which is separate from the actual transport technologies involved. In contrast, DTS is a web service implementation, which integrates encryption as part of the transport via SSL (https).

The encryption standards (OpenPGP and SSL) are implemented by products from multiple software vendors which provide competing solutions using the same interoperable encryption standards. Having multiple vendors from which to select an encryption product allows each organization participating in the CommonLine, CAM, or CR:C file exchange process to choose the encryption product that best meets its specific needs.

SSL Encryption

Organizations implementing a DTS Service must obtain an X.509 certificate suitable for the site's web server, thus enabling secure communications. The certificate must be obtained from a recognized certificate authority such as VeriSign® or a similar organization. Each organization should consult its web server's documentation (or Technical Support staff) for details on obtaining and installing an X.509 certificate.

NOTE

Test certificates or self-signed certificates must not be used in production environments.

Communication using SSL (https) works as follows:

- The (DTS) client connects to the web server and as part of the normal connection process (*"handshake"*) asks for the certificate.
- The web server returns the certificate to the client.
- The client then determines which certificate authority authorized and signed the certificate and verifies that the signature is valid by using the public key for that authority.
- If the certificate is found to be valid, the two computers begin communication with encryption.
- If the certificate, domain name, or keys do not match up correctly, the connection is aborted.

While this is an over simplification of the SSL encryption protocol, there are two key items to point out. First, all of these steps are part of the standard https protocol and lie completely outside of a DTS implementation. Second, the public key exchange occurs during the "handshake" of client and server, so there is no need for manual exchange of the public key contained in the X.509 certificate. Commonly available Web Server applications and Web Browser applications have an SSL implementation built-in.

OpenPGP Encryption

The OpenPGP encryption standard is defined in Internet Engineering Task Force (IETF) RFC 2440. Since OpenPGP is an open standard, there are multiple implementations. GNU Privacy Guard (GPG) and PGP Corporation's software are two implementations that are available today and have been tested by the NCHELP Electronic Exchange Advisory Team. Since there are

multiple implementations of the OpenPGP standard, organizations should select encryption products that are certified by the products' vendors to be OpenPGP compliant.

Additional information regarding GPG can be found at the GNU Privacy Guard Website at *http://www.gnupg.org/*. GPG is a fully open source project. It is installed by default on many GNU/Linux systems but can also be installed on FreeBSD, OpenBSD, and Windows and can be compiled to run on many other platforms as well. A current list of compatible platforms can be found on the GNU Privacy Guard Website.

Additional information regarding PGP Corporation's PGP software can be found at *http://www.pgp.com/*. This Web site offers a full service product line and technical support. Versions of the PGP software are available for Macintosh, Windows, UNIX, and OS/390 systems.

Communication using OpenPGP systems for public key encryption works as follows:

- Prior to sending the message, the sending organization obtains a copy of the receiving organization's public key; this key is added to the sending organization's database of public keys (the key ring). For CommonLine, CAM, and CR:C processing, each organization will have a key ring containing the public keys of all of its trading partners.

- The sending organization encrypts the message using the receiving organization's public key, signs the message with the sending organization's private key, and sends the message to the receiving organization.

- The receiving organization decrypts the message using its secure private key, and verifies the received message's signature with the public key of the sending organization.

See diagrams, PGP Encryption & Signing and PGP Decryption & Signature Verification, below for illustrations showing OpenPGP encryption.

Digital Signatures

A secondary element of the CommonLine/CAM/CR:C secure transmission process is the use of a digital signature within the message. The digital signature protects against a third party impersonating the sending organization or tampering with the message. Digital signatures use a variant of the public key scheme. While encryption is performed using the receiving organization's public key, digital signatures are performed using the sending organization's private key. The digital signature is encrypted using the sending organization's private key and is decrypted and verified by the receiving organization using the sending organization's public key. The digital signature itself is a string of text, which, among other things, contains a compact mathematical digest of the content. If the text is altered in transit, the digest will no longer correspond to the received data, and the digital signature verification process will fail. If the receiving organization's software can successfully read the signature, the receiving organization can be confident the message is authentic.

See the APPENDIX A: OPENPGP SPECIFICS for examples /directions for creating a digital signatures for use with the FTP or POP/SMTP transport protocols. Refer to Diagram 1 and Diagram 2.

The PESC DTSv1 Specification gives specific implementation requirements for Digital Signatures. A certificate (X.509) signed by a Certificate Authority (CA) is required. The DTSv1 Reference Implementation guide also provides more information on creating the signature. Refer to Diagram 3 and Diagram 4.

Diagrams

Diagram 1: PGP Encryption & Signing

The following diagram shows Alice sending encrypted and signed data to Bob.

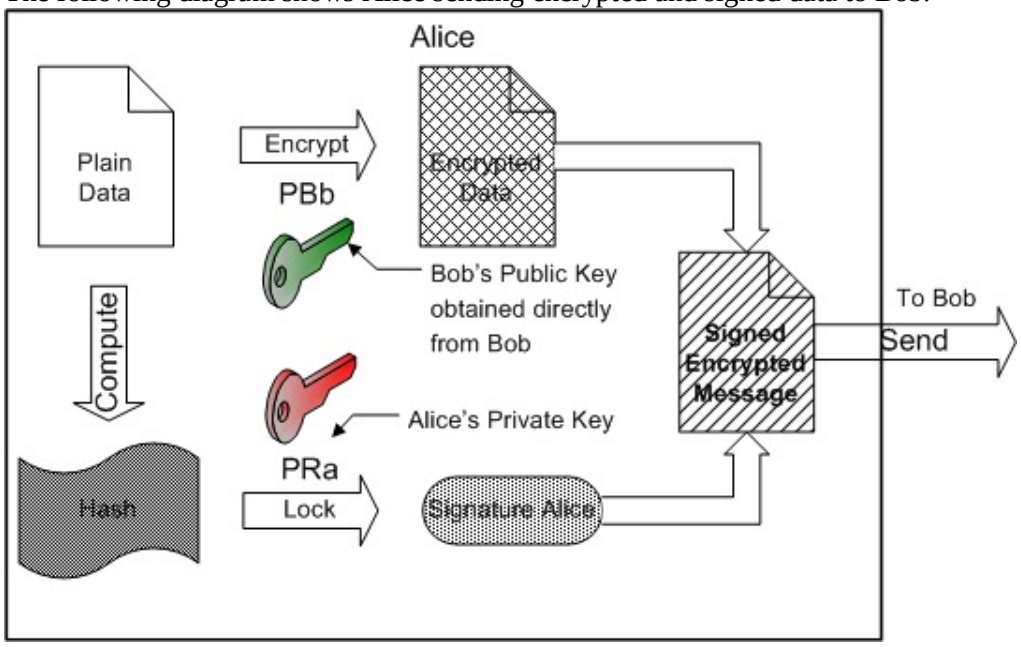


Diagram 2: PGP Decryption & Signature Verification

The following diagram shows Bob decrypting and verifying the signature for a message received from Alice.

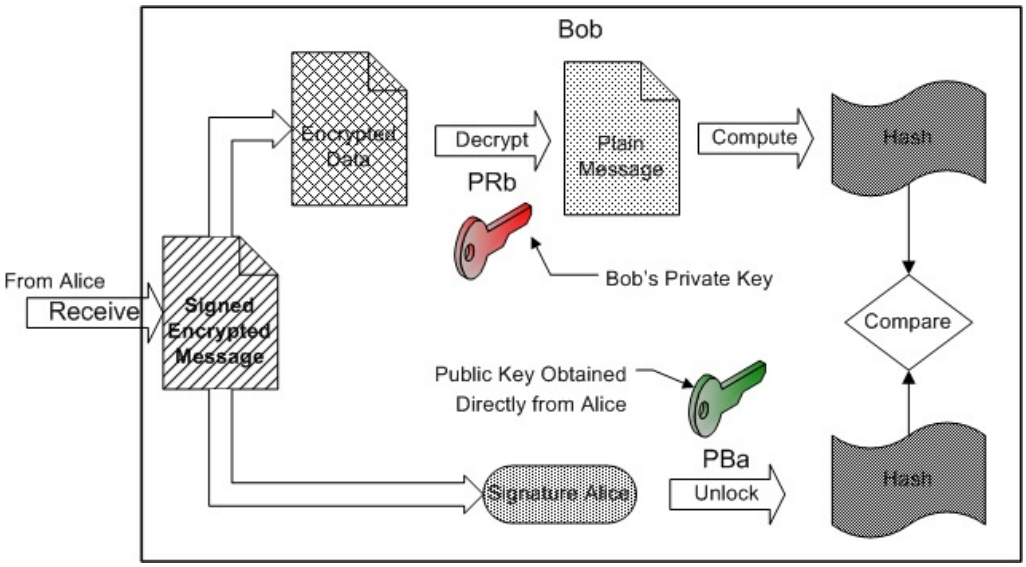


Diagram 3: DTS v1.0.1 - Digital Signature

The following diagram illustrates the use of Alice's private key to sign the payload of a DTS v1.0.1 message. See the PESC Data Transport Standard Specification document and the Data Transport Standard section of this document for more information regarding creating the DTS v1.0.1 SOAP Package.

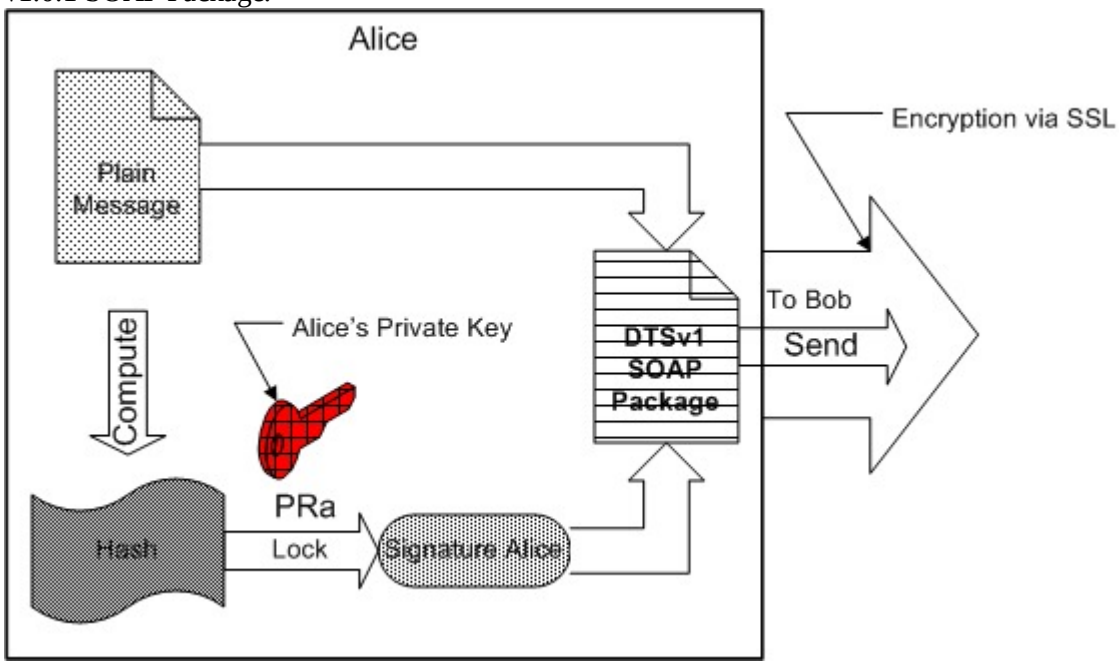
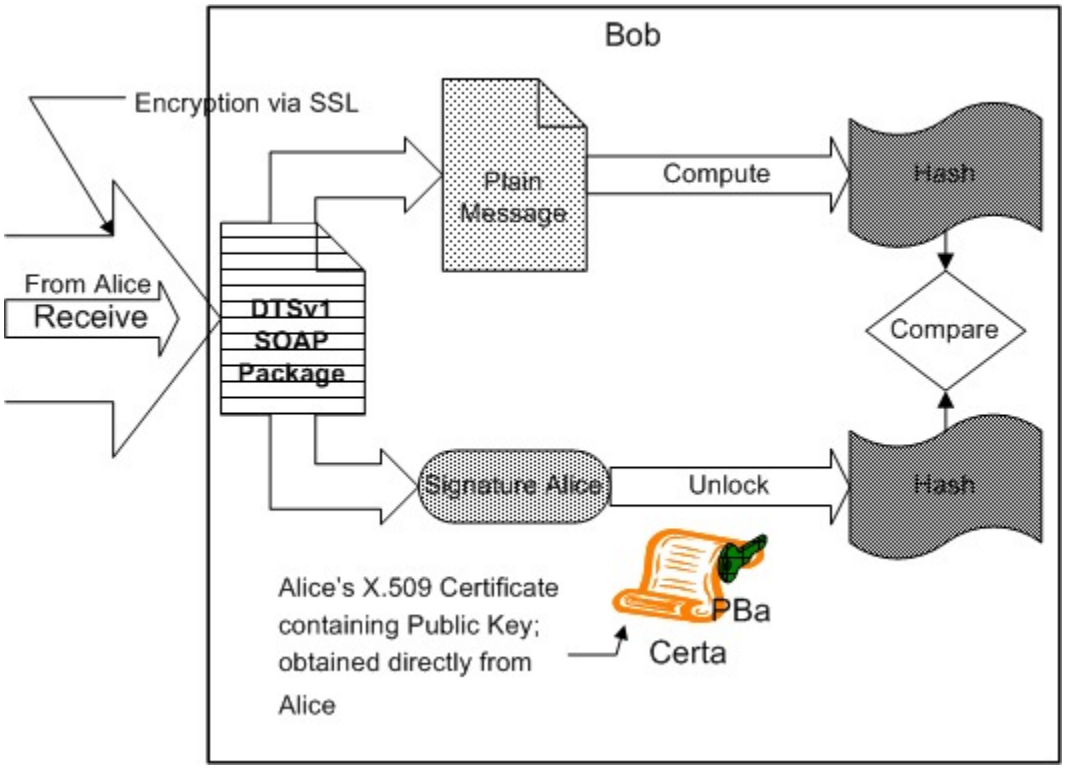


Diagram 4: DTS v1.0.1 – Verifying Digital Signature using X.509 Certificate

The following diagram illustrates the use of Alice's X.509 certificate to verify the signature of the payload in a DTS v1.0.1 message. See the PESC Data Transport Standard Specification document and the Data Transport Standard section of this document for more information regarding creating the DTSv1 SOAP Package.



THIS PAGE IS INTENTIONALLY BLANK

Security and Authentication: Key Management

Key Generation

When an organization uses encryption for CommonLine/CAM/CR:C data files for the first time, the initial task is to generate a public and private key pair. Copies of both public and private keys should be saved to external media and stored off-site in a secure location.

OpenPGP

Standards for encryption using OpenPGP as outlined in RFC2440 are flexible and offer wide latitude in the selection of ciphers, asymmetric key characteristics, signature methods and hash (or digest) methods. As cryptanalysis techniques become more sophisticated, adequate security relies (in part) on choosing key characteristics of sufficient strength to protect against mathematical compromise. It is left to each file trading organization to choose key characteristics that provide ample defense.

However, freely choosing any of the available methods may lead to incompatibility between trading partners. Therefore the following characteristics are strongly recommended for keys used within the Electronic Exchange community:

Cipher:

- 3DES
- AES (128, 192 and/or 256 bit)

Public Key:

- DH - including ElGamal (2048 bit Key length MINIMUM)
- RSA (2048 bit Key length MINIMUM)

Signature:

- DSA – Including DSS
- RSA

Digest

- SHA-1 and SHA-2 Family

Compression

- ZIP compression as referenced in RFC2440 (defined in RFC1951)

Since each trading partner may select any of the above methods when assembling a key pair or configuring PGP it is important that each partner in the trading community accept and allow ALL of the above methods. Proprietary algorithms such as IDEA are not in themselves completely “open” and thus are strongly discouraged. There is no guarantee that a trading partner will be able to accept files encrypted with such methods.

Also, algorithms that are known to be less secure or that have demonstrated weakness in the cryptanalysis community should be avoided including MD5 (Digest) and CAST5 (Cipher).

There are other encryption algorithms such as Twofish and Blowfish that are known to be secure, however to maintain satisfactory compatibility it is recommended that only the above listed methods be used in the trading community.

It is worth noting that the US Federal Government Standards for encryption (Federal Information Processing Standards) are published by NIST (National Institute of Standards and Technology). These standards recommend the use of only 3DES or AES for cipher methods and DH or RSA for Public Key methods.

Key Identifier Specifics

The key identifier value (X.Y.Z) in the e-mail subject line/FTP filename must exactly match the OpenPGP User ID value (X.Y.Z) contained in the sending organization's public key, regardless of the contents of the file being transmitted. These key identifier values from the e-mail subject line/FTP filename indicate the private key used to sign the message and the public key needed to verify the signature.

When generating keys a user id must be specified. For EEAT communications, a user id with the following format, which is similar to the key identifier used in an e-mail subject line or an FTP filename, is required. Spaces are required after the B component and before the "<" for the E component in the OpenPGP user ID.

The OpenPGP user ID components are as follows:

B (A.X.Y.Z) <E> where:

B = The entity name, which is composed only of alphanumeric characters or spaces. This value is required.

A = The file type (CLDATA), which identifies the key as being for CommonLine, CAM, or CR:C files. This value is required.

X.Y.Z = The key identifier, which is comprised of up to three segments that identify the sending organization for each file transmitted. The key identifier portion of the sending organization's user ID must be identical to the key identifier portion of the sending organization's s e-mail subject line, FTP filename, or appropriate SOAP Header elements for DTS.

The individual segments of the key identifier are as follows:

X = A constant value **DOE or VID**, which must be in uppercase characters.

This value is required.

NOTE

The **X** field will contain “DOE” for both the NCHELP identification number and the ED identification number; for organizations that have one. For organizations that do not have a DOE identification number (NCHELP or ED) a Vender Identification number, “VID”, must be used.

Y = The unique identification number assigned by ED or NCHELP.

ED - Numeric ID values are assigned by ED (i.e.; 3-digit ID’s to guarantors, 6-digit ID’s to lenders and servicers, and 8-digit ID’s to schools).

NCHELP - Alphanumeric "3-8 character" ID values are assigned by NCHELP. Only servicers will be assigned this type of ID number. The length of this non-ED assigned identifier is not related to the type of servicer to which it is assigned.

This value is required.

NOTE

The NCHELP identification number will be different than the identification number assigned by ED. Zeros and spaces must not be used as padding. Only zeros that are part of the actual ED identification number may be used.

Z = The non-ED branch ID. This value may be comprised of up to four alphanumeric characters. If the non-ED branch ID is not used, there will be no delimiting period (.) following the **Y** value. This value is optional unless it is needed for proper routing and delivery.

NOTE

Trading partner systems will be required to differentiate between organization non-ED branch IDs transmitted as part of the information in the e-mail subject line or FTP filename. In addition, zeros and spaces must not be used as padding. Only zeros that are part of the actual non-ED branch ID may be used.

The file type and key identifier components of the OpenPGP user ID must be enclosed in parentheses, and there must be no spaces.

<E> = An optional default e-mail address enclosed in angle brackets (< >).

For additional instructions and option settings required for interoperability with EEAT, refer to APPENDIX A: OPENPGP SPECIFICS.

X.509 Certificates

Standards for encryption using X.509v3 certificates are outlined in the IETF Public-Key Infrastructure (X.509) (pkix). The EEAT partner creates a public-private key pair on their own computer using commonly available software, specific to their operating platform. The public key is sent to the certificate authority and the authority returns a signed X.509 Certificate. An X.509 Certificate is a container for a public key. In addition, the certificate is digitally signed by the issuer of the certificate, a trusted certificate authority. Therefore, the X.509 Certificate may be distributed to anyone. Since the certificate contains only public information, it need not be encrypted before transmission.

NOTE

It is possible, and in some circumstances may be necessary, to embed both the public and private key in an X.509 certificate. If this certificate is shared or lost, it will result in the compromising of the security of the private key which would require the re-generation and re-signing of the key by the certificate authority. Completing this process can take several days and would represent a significant business disruption. **DON'T EXPORT YOUR PRIVATE KEY, UNLESS YOU ARE AWARE OF ALL THE POTENTIAL REPERCUSSIONS.**

The private/public keys and the associated X.509 certificate have two uses when implementing the DTS transport protocol.

1. To secure the web site (and DTS Service) under SSL encryption (https).
   (network administrators typically deal with obtaining the certificate, see SSL Encryption)
2. To authenticate the sender via digital signatures.

For implementing digital signatures the program handling transport must have access to the X.509 certificate.

With two uses of an X.509 Certificate, there is the possibility to have two distinct Certificates per organization; both must be obtained from a Certificate Authority (see Diagram 3). If a DTS Client implementation (see Transport Protocols: DTS) has access to the X.509 certificate that is installed on the web server then the same certificate may be used by the client for digital signatures as is used by the service for both purposes.

The X.509 Certificate that is used for Digital Signing is the only certificate that is required to be exchanged for DTS v1.0.1. Each organization will therefore have only one digital certificate per trading partner in their key store. The trading partner does not need the X.509 Certificate being used for SSL since the https protocol handles the key exchange, see SSL Encryption section.

There must only be one X.509 Certificate that is used for Digital Signatures in DTS, per organization ID plus non-ED Branch ID as defined by ED or NCHELP. If an organization is running both a DTS Service (SSL enabled) and a DTS Client, the same certificate must be used for creating the signatures. If the certificate to secure the service is used to sign the responses then that same certificate must be used by the client to sign the requests.

The OpenPGP specification allows the explicit assignment of the User ID that is created as the key identifier for transport and this document states the rules for assigning the value (see key identifier specifics). However, the X.509 Certificate obtained from a Certificate Authority has more explicit rules that can not be modified by this standard. The implementation of PESC DTS v1.0.1 Transport protocol by NCHELP-EEAT defines values to be used as the same type of key identifier specifics that are used with OpenPGP; these values are communicated in SOAP Header elements and are defined in the DTSRequestRouting and DTSResponseRouting Header Elements section.

Since the values transmitted as the key identifier in the DTS Header are not the same as what will be used to pull a certificate from the certificate store, there also needs to be a mechanism employed to associate the values transmitted as the key identifier to the “Subject” or “CN” of the X.509 Certificate (see Database of Public Keys / Key Ring).

Where applicable, the key pair must be able to support SSL 128 or better. In some cases, depending on local laws, this will be unacceptable. The EEAT recognizes this situation and recommends that trading partners accommodate such restrictions.

The requirement of obtaining the X.509 Certificate from a trusted Certificate Authority is to allow the authenticity of the certificate to be independently validated (see CA Signing Process).

Key exchange

Once the key pair is generated, public keys must be exchanged with all trading partners.

OpenPGP

Follow these steps to exchange an OpenPGP public key with a trading partner (see Diagram 1):

1. Generate the public and private key pair using the appropriate encryption standards and the user ID (see OpenPGP- Key Identifier Specifics).

2. Export the public key to a file, if the public key is combined with other public keys on a key ring.

3. Transmit the public key using the CL COMM UPDATE file type in the e-mail subject line or the CL_COMM_UPDATE file type in the FTP filename (see Valid Values - File Type Component and E-mail subject line components or FTP filename components). When transmitting public keys for the first time, sending organizations may choose to send a single message to all trading partners.

4. The receiving organization must contact the sending organization to determine the validity of the public key update (see OpenPGP - Verifying public key exchanges). Only a valid public key update, confirmed out-of-band with the sending organization may be processed.

5. If the receiving organization previously had a public key for the sending organization, the receiving organization will replace the existing public key with the newly received and verified public key. If the receiving organization did **not** have a public key for the sending organization, the receiving organization will add the newly received and verified public key to its public key ring.

Steps 2-5 must be repeated each time an organization adds a trading partner. It is important to verify that newly received public keys actually come from the correct source. Not doing so may compromise the security of the system (see OpenPGP - Verifying public key exchanges).

X.509 Certificate

Follow these steps to exchange X.509 Certificates with trading partners (see Diagram 2)

1. Generate the public and private key pair using available software, specific to operating platform.

2. Perform the Certificate Request process with a trusted Certificate Authority to obtain a CA-signed X.509 Certificate (see Diagram 3).

3. Transmit the Certificate:
   a. Transmit the certificate using the CL COMM UPDATE file type in the e-mail subject line or the CL_COMM_UPDATE file type in the FTP filename (see Valid Values - File Type Component and E-mail subject line components or FTP filename components). When transmitting public keys for the first time, sending organizations may choose to send a single message to all trading partners.

   b. Transmit the certificate using the CL COMM UPDATE PayloadType Header in DTS (see Valid Values -Payload Type Header). DTS requires a single point-to-point transmission for each trading partner to receive the key.

NOTE

The DTSv1 Specification requires creating and validating the digital signature of the payload sent. While it is possible to send the certificate as the payload, the receiving system would need to be capable of extracting the certificate, verifying its authenticity and then using it to verify the authenticity of the original payload (certificate). The recommendation of the EEAT is to use POP/SMTP or FTP to exchange a certificate for the first time due to the complexities of the process noted.

4. The receiving organization must verify the authenticity of the certificate (see CA siging process). Only a valid and authentic certificate may be processed.

5. If the receiving organization previously had a certificate for the sending organization, the receiving organization will replace the existing certificate with the newly received and authenticated certificate. If the receiving organization did **not** have a certificate for the sending organization, the receiving organization will add the newly received and verified certificate to its certificate store.

Steps 2-5 must be repeated each time an organization adds a trading partner. It is important to verify that newly received certificates actually come from the correct source and are issued by a trusted Certificate Authority. Not doing so may compromise the security of the system (see X.509 - Verifying certificate exchanges).

Verifying public key/certificate exchanges

The current ESC rules governing public key or certificate exchanges specify that they must be verified upon receipt. The verification process must be external to the method by which the key or certificate was received -- this is called an "out-of-band" process. Both specifications for the keys employed have mechanisms in place to obtain the digital fingerprint of the key. X.509 Certificates, by virtue of the Certificate Authority requirement, has an additional mechanism where the certificate is authenticated by an external third party.

OpenPGP

To verify the validity of a key received the receiving organization will contact the sending organization "out-of-band" from how the key was received. During this communication, the fingerprint of the key received will be provided to the sending organization so that they can confirm that the correct, current key arrived intact and unmodified. If the two fingerprints match exactly, the public key has not been altered. If the fingerprints do not match, there has been a problem with the public key exchange. This communication will take place through a previously described channel.

NOTE

The DTS implementation defined by this document provides an optional and acceptable "out-of-band" verification process; see Verification out of band via HTTPS (optional).

The risks of public key exchanges and the challenge of troubleshooting problems can be minimized through the use of fingerprints. Questions regarding this process should be submitted through the ESC online issue submittal process via *http://www.nchelp.org*

X.509 Certificate

There are multiple ways for verifying the validity of an X.509 Certificate. After the key is processed, a finger print/thumbprint can be obtained and verified with the sending institution, just as with the OpenPGP Key verification process. By using the X.509 Certificate as the container

for the public key, it also contains additional information including the signature of the public key from a trusted third party, Certificate Authority, and a "Trust Chain" that allows all signatures of a key to be validated up to a Root Certificate that the receiving organization trusts as well.

An X.509 Certificate must be issued by and obtained from an accepted third party Certificate Authority. While it could be a good practice to contact the sending institution "out-of-band" to verify an update was sent, the CA requirement and verification steps possible from it will be the standard for certificate verification. Additional steps can be added by mutual agreement between two trading partners.

Certificate Authority signing process

An X.509 Certificate is a container for a public key. In addition, the certificate is digitally signed by the issuer of the certificate, a trusted Certificate Authority (CA), and that signature is also in the certificate. The public key of most trusted CAs already exist on most computers. This allows the validation of the CA's signature of an organization's public key (both within the certificate) without additional key exchange. The X.509 certificate is trusted, by the assurance that the CA has adequately validated the identity of the certificate's owner (exact means and liability stipulated by the CA's contract). There can be multiple levels of a certificate and subsequent issuer that make up a chain of trust. In addition to these most critical components of a certificate, certificates also contain a means by which to utilize a revocation list controlled by the CA.

Upon receiving a new certificate from a trading partner, the recipient is responsible for verifying that the certificate was issued by a Certificate Authority they trust and that it has not been revoked by that authority. This will vary based on operating system, platform, and tools; contact your system administrators for more details for implementing this practice.

Verification out of band via HTTPS (optional)

The receiving organization must verify receipt of a public key with the sending organization outside of the method in which it was sent, thus "out-of-band". Currently, this "out-of-band" mechanism is a phone call to a number previously disclosed to the trading partner during the initial negotiation/set up period for electronic trade, where the fingerprints on each side are compared for equality. The primary means of trading keys is POP/SMTP and FTP, thereby making the phone call a valid "out-of-band" mechanism.

During the initial negotiations and set up processes, it is considered valid for a web address protected by SSL (https) to be given in addition to a contact phone number for key verification. Since the web address is only given at this time and it will not be contained in any actual key exchanges, it is another way for verifying the key "out-of-band."

Upon receiving a transmission identified as a key exchange, the recipient should pull it out of the normal processing stream for additional and often manual processing. The fingerprint of the received key is obtained and that fingerprint is then compared to the fingerprint of the sending organization's current key.

The following two sections will describe how fingerprint comparisons can occur via HTTPS either automatically (implementing DTS) or one-sided manually (web page). Since it relies on certain technical implementations by trading partners, these should be considered optional and valid based only on a mutual trading partner agreement.

DTS

If the entire communication process is automated, the fingerprint of the newly received key should have been obtained and set aside for processing. This section assumes that the sender and recipient of the key are fully automated and that they have implemented the DTS transport protocol.

An automated incoming process follows these steps:

1. Once the transmission is identified as containing an OpenPGP key, the recipient computes the key's fingerprint.
2. The recipient of the key obtains the DTS Service End-point from an internal repository for verifying the key using the entity and key identifier information contained in the transmission.
3. The recipient of the key uses the DTSRequestPayloadType of "Key Verification 02" with the fingerprint as the payload.
4. The DTS Service extracts the fingerprint from the DTS Request and compares it to the fingerprint of their current key, held internally on their system.
5. The DTS Service responds using the DTSResponsePayloadType of "Key Verification 02 Response":
    a. If the fingerprint sent by the DTS Client matches what is internally held, the payload is "True."
    b. If the fingerprint sent by the DTS Client does not match what is internally held, the payload is "False."
6. The DTS Client processes the response from the service:
    a. If "True" is returned; the recipient of the key (DTS Client) is justified to insert the key received into their key ring/database.
    b. If "False" is returned; the recipient's system should immediately notify their internal staff which should then immediately contact the sending institution.

Web Page

This method of verifying the public key is parallel to the existing method via a phone call.

These steps would be followed when the organization that receives the new key must verify it manually:

1. Existing process would be followed up to point of finding the contact phone number of the organization that sent the key update.
2. The user at the recipient organization would look up the web site (https) and enter the URL into a browser
3. The web page displayed to the recipient would have a single text box allowing entry of the fingerprint (this text box should be implemented with a maximum length of 50, the length of a valid fingerprint of a PGP key).
4. The user at the recipient organization would take the fingerprint obtained by some existing mechanism and insert it into the text box provided on the web page (instead of reading over the phone)
5. The web site would take the value entered and compare it to the fingerprint of their current key, held internally on their system.
6. The web site would return a web page:
    a. If the fingerprint entered by the recipient matches, a confirmation of valid key fingerprint would be displayed

   b. If the fingerprint entered by the recipient is not the same, an invalid message would be displayed
7. The user would:
   a. Upon confirmation of valid fingerprint, follow existing steps allowing a valid key to be used by the recipient organization
   b. Upon invalid key notice, immediately look up the contact phone number and contact the sending organization.

Database of Public Keys / Key Ring

As public keys are received, they should be added to a public key database, also known more commonly as the key ring (Diagram 1 and/or Diagram 2). It is recommended that organizations back up their key rings after each public key update. Consult your technical support group for key ring back-up procedures.

There must also be a means by which to associate a public key or certificate with a particular organization derived from information readily available in the transmission. Implementing an OpenPGP "Key Ring" and following the requirements described in OpenPGP - Key Identifier Specifics section, the "Key Ring" itself provides this ability.

However, there is additional information that should also be tied to an organization that is necessary for transport. This additional information includes, but is not limited to, transport protocol supported, address for receiving data, X.509 Certificate "Subject" or "CN", and even contact information. Such additional information leads to the recommendation of a database/table of public keys and trading partner information. The ESC approved standards requires the use of an "X.Y.Z" key and sender identification construct that is implemented in all transport protocols and thus makes an excellent choice for primary key or look up criteria.

At a minimum a trading partner must:

- Be able to send data with the appropriate encryption/digital signing, transport protocol, and "address" for an entity on a per file type basis
- Be able to retrieve the public key of a partner based on the "X.Y.Z" construct based on encryption protocol of a single transmission

Updating keys

There may be occasions when a new public and private key pair must be generated by one of the trading partners. Updates can be necessitated by a lost pass-phrase, a security breach, or internal security protocols that dictate the periodic update of keys. In all cases, when updating the key ring, be sure to make back-up copies. The sending organization must determine the timing of public key updates. Optimally, public key updates should be coordinated when processing data files to minimize delays. When updating public keys, sending organizations may wish to name the transport file to indicate the public key owner. The recipient of a public key update is not required and is not expected to use the filename provided by the sending organization when saving the public key.

NOTE

Organizations should understand that frequent key updates might strain the resources of their trading partners. Such strains can reduce the overall efficiency of file transmissions. Acknowledging this and creating a practice of updating keys once or twice a year with adequate security for the private key should suffice. Additionally, key updates require extensive time-sensitive coordination among trading partners. It is strongly recommended that older keys be maintained for

non-repudiation and troubleshooting of older transmitted data.

Follow these steps to transmit a public key update:

1. If a security breach has occurred, contact all trading partners to suspend data file transmissions until security is restored.

2. Follow steps 1-5 in *Key Exchange OpenPGP* or Key Exchange X.509 Certificate (for more general information see Key Generation)

3. Identify any data file transmissions that are in-progress. Re-encrypt such data files with the new public key, and re-transmit the data files.

The previously described steps must be repeated each time a public key update is needed. It is important to verify that newly received public keys actually come from the correct source. Not doing so may compromise the security of the system. To verify that a public key came from the correct source, the receiving organization must contact the sending organization out-of-band. E-mail and FTP are not accepted as a secure method for verifying the origin of a public key update.

Deleting keys

If an organization decides to cease participation in the CommonLine/CAM/CR:C data file transmission process, trading partners must delete that organization's public key from their key rings.

Follow these steps to remove public keys:

1. The organization departing the business network must verify that all final data file transmissions have been received and processed.

2. The organization departing the business network must notify each trading partner out-of-band to delete their public key from the trading partner's key ring.

Diagrams

Diagram 1: Exchanging OpenPGP format Public Keys

The following diagram illustrates the key exchange process:

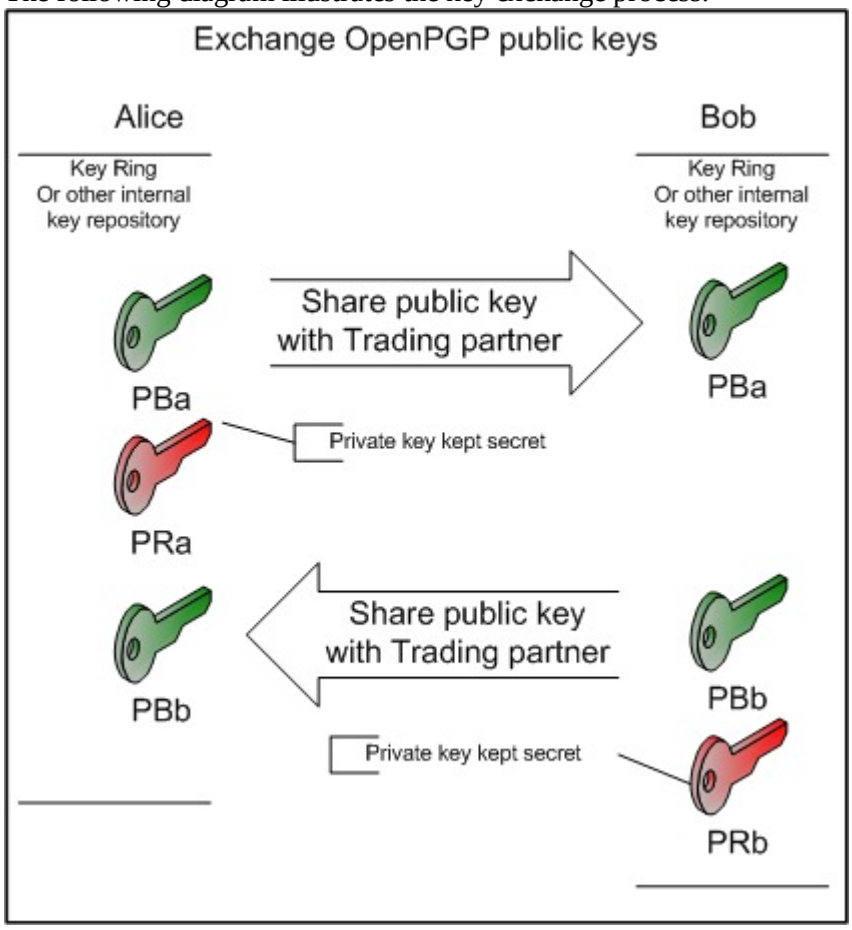


Diagram 2: Exchanging X.509 certificates

The following diagram illustrates the X.509 certificate exchange process:

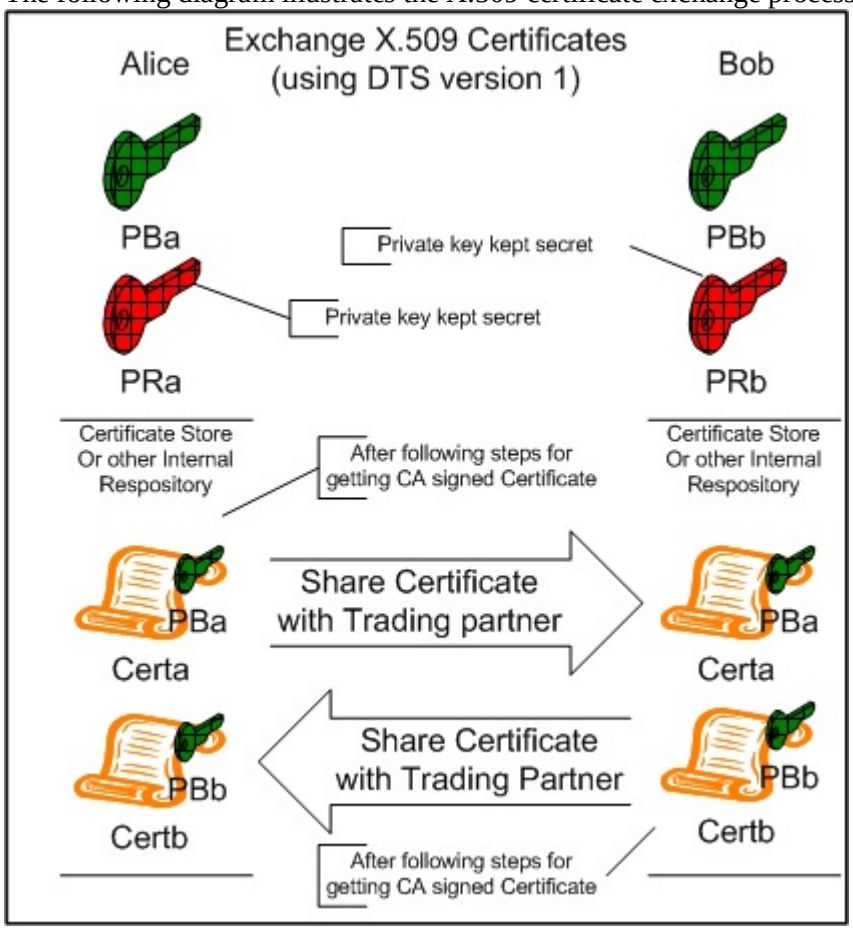
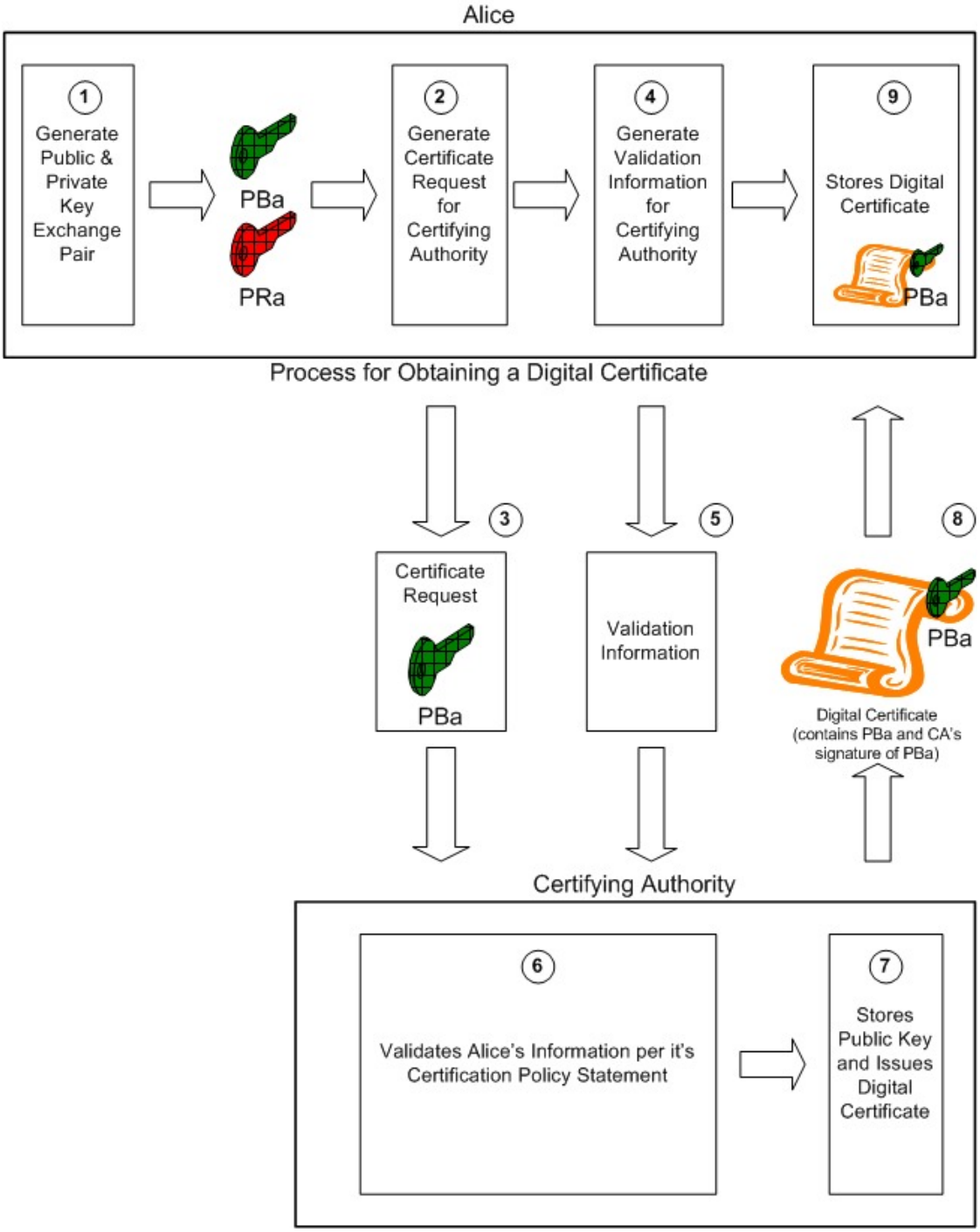


Diagram 3: Obtaining Certificate Authority signed X.509 Certificate

The following diagrams illustrates the abstract method of obtaining an X.509 certificate that is suitable for creating digital signatures used for the Data Transport Standard:



TRANSPORT PROTOCOLS: SMTP/POP3

Introduction

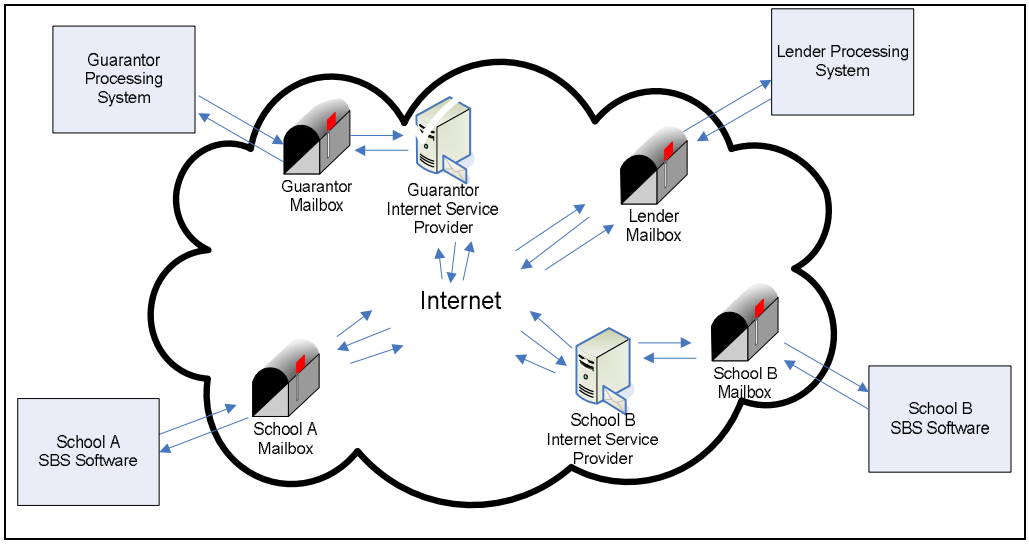
The purpose of this section is to provide CommonLine, CAM and CR:C participants the procedures necessary for using an Internet-compliant Simple Mail Transfer Protocol /Post Office Protocol 3 (SMTP/POP3) system.

SMTP is a protocol for message-based transmission of files and is optimal for file attachments that are less than 1 MB in size (post encryption).

RFCs 821, 822, 1869, and 1870 define the SMTP transport protocol.

Conceptual overview

The following diagram illustrates data files exchanged between organizations using SMTP/POP3.



It is mandatory that all organizations using the SMTP/POP3 transport protocol encrypt and digitally sign their data file attachments. Secure data encryption is necessary to provide information privacy for CommonLine, CAM, and CR:C data files. By encrypting a data file prior to transmission, the data file is locked by the sending organization and must be later unlocked by the receiving organization. Locking and unlocking is accomplished using encryption keys. In addition to locking and unlocking the data file, encryption can provide the sending organization's digital signature. The digital signature confirms to the receiving organization that the authorized sending organization actually transmitted the data file.

SMTP/POP3 business partner responsibilities

An organization's system changes must be in place before its trading partners can begin using SMTP/POP3 to transmit information. Each trading partner must establish an e-mail account that is SMTP/POP3 compliant. An organization can accomplish this by establishing an e-mail account through a local or national Internet Service Provider (ISP). CommonLine, CAM, or CR:C trading partners who have a full time connection to the Internet may choose to install their own SMTP/POP3 e-mail servers.

In addition to the Internet e-mail account, an organization will need to generate a public/private key pair and exchange public keys with its trading partners. Public and private keys are used to encrypt, decrypt, digitally sign, and authenticate data files transmitted via the Internet. (See the Security and Authentication: Key Management section for a detailed explanation of encryption and public/private keys.) An organization must communicate both its e-mail address and its public key to its trading partners.

Follow these steps to convert to the SMTP/POP3 transport protocol:

1. Establish the public and private encryption key pair, using the appropriate encryption standards and user ID. See Key Identifier Specifics section.

2. Establish an Internet mailbox for incoming e-mail. See ***E-mail system requirements*** in this section.

3. Create a process for exchanging public keys with CommonLine, CAM, and CR:C trading partners that will do the following:

    - Send the public key to trading partners.

    - Record trading partners' public keys.

    - Verify public key updates with the sending organization.

    - See Key Exchange section.

4. Establish a database of public keys (a key ring) for trading partners. See Database of Public Keys / Key Ring section.

5. Create processes that will do the following to handle incoming files:

    - Retrieve the incoming e-mail from a POP3 mail drop—a directory populated by an SMTP server—or an e-mail package

    - Decrypt (unlock) the incoming data files.

    - Verify the digital signatures of the incoming data files using the sending organization's public key.

NOTE

For school-based software vendors, the two previous bullets may appear as a single step.

- Move the information to existing internal programs for processing.
- See the Security and Authentication: Mechanics section for more information.

6. Create processes to handle outgoing files that will do the following:
   - Encrypt (lock) the outgoing data file with the receiving organization's public key.
   - Add a digital signature using the sending organization's appropriate private key.
   - Send the data file to the receiving organization's e-mail address via SMTP.
   - See the Security and Authentication: Mechanics section for more information.
7. Sending organizations should retain a local copy of all transmitted data files in case retransmissions are necessary. Receiving organizations should log incoming data files to permit better tracking of data file exchanges.

SMTP/POP3 SBS vendor responsibilities

School-based software (SBS) vendors must ensure that appropriate updates are made to the SBS before files are exchanged. SBS vendors must add functionality to enable SBS products to perform various encryption processes. For additional SBS information, reference the ***General Guidelines for SBS Development*** section.

Follow these steps to develop Internet communications in SBS software:

1. Add an encryption key generation method to the SBS to enable the school to generate a public and private key pair using the appropriate key identifier. The private key should be secured.
2. Add an interface to the school's established Internet mailbox using SMTP/POP3.
3. Establish a database of public keys (a key ring) for the school and its trading partners. See Database of Public Keys / Key Ring section.
4. Create processes to enable the exchange of public keys between the school and its trading partners. These processes must make it possible for SBS users to be aware of and verify incoming public key update transactions prior to using the updated public key.
5. Provide processes that automatically encrypt/decrypt data files being transmitted.
6. Assist customers with setup and configuration details as needed for encryption and file transport.

E-mail system requirements

Whether an organization uses its own internal e-mail system or selects an ISP, it is critical that the e-mail system meets the following requirements:

- It must be fully SMTP compliant.
- It must be accessible by any POP3 compliant e-mail software.
- It must be capable of sending and receiving files using the MIME (multipurpose Internet mail extensions) standard with BASE64 encoding.
- It must support the transmission of a file up to 1 MB in size (post encryption)
- It must provide enough file storage space to accommodate business needs.
- It must provide a mailbox for CommonLine, CAM, and CR:C files. It is strongly recommended that provisions be made for a dedicated mailbox to be used exclusively for these files.
- It must be able to support a 70 character e-mail subject line.

E-mail subject line components

The e-mail subject line provides a variety of information to the receiving organization. The maximum length of an e-mail subject line is 70 characters. The A, B, and left angle bracket (<) values are separated by a space. These values are required.

The e-mail subject line components are as follows:

A [B] <X.Y.Z:M> where:

A	=	The file type.
B	=	The encryption protocol code.
X.Y.Z	=	The key identifier.
M	=	The unique message identifier.

File Types (A)

The first component of the e-mail subject line is the file type (component "A" in E-mail subject line components). The file type identifies the specific type of file being transmitted. The file type must always occur as the first component of the e-mail subject line to enable the receiving organization to sort by file type. For a list of valid values for this component of the subject line see File Type Component and Payload Type Header table in the Valid Values for Components and Examples section.

NOTE

Embedded spaces are part of some file types.

Encryption protocol code (B)

The second component of the e-mail subject line is the encryption protocol code (component B in E-mail subject line components). The encryption protocol code identifies the type of key and algorithms used to encrypt and digitally sign the data file. The 2-digit encryption protocol code is mandatory and must be enclosed in square brackets ([]). See Encryption Protocol Code in General Encryption Section. For a list of valid values for this component of the subject line see the Encryption Protocol Code table in the Valid Values for Components and Examples section.

Key identifier (X.Y.Z)

The third component of the subject line is the key identifier (component X.Y.Z in E-mail subject line components). The key identifier is subdivided into three segments that identify the sending organization. It is mandatory to separate the file type and encryption protocol code from the key identifier with a space in the e-mail subject line. The key identifier in the e-mail subject line must exactly match the key identifier contained in the user ID of the public key regardless of the contents of the file being transmitted. See Key Identifier Specifics in the Encryption section for more detailed information on the values of the X.Y.Z components of the subject line.

The support of each component is mandatory.

Unique message identifier (M)

The fourth component of the e-mail subject line is the unique message identifier (component M in E-mail subject line components). The unique message identifier enables trading partners to uniquely identify a message to facilitate file tracking. In addition, the unique message identifier may be used to match the acknowledgment back to its original message. Only uppercase alphanumeric characters (A-Z and 0-9) and the period (.) may be used in the unique message identifier. This message identifier must be unique for at least 90 days.

The unique message identifier must be is preceded by a colon (:) and is composed as follows:

M where:

M = The unique message identifier. This component of the e-mail subject line is required.

NOTE

It is strongly recommended that this value be derived from a date/time stamp in the format of "CCYYMMDDHHMMSSnnnnnnnn". No time-zone identifier is included in this scheme. It is assumed that the time stamp is relative to the server time-zone that created the data transmission. Utilizing this format for the "M" section will simplify the compliance with the "90 day" uniqueness requirement, actually making it unique forever.

The key identifier (described in the previous section) and the unique message identifier components of the e-mail subject line are enclosed in angle brackets (< >) and must not exceed 41 characters (including the angle brackets).

Refer to the Examples - Email Subject Line section to be shown how all these elements are put together.

E-mail body

It is strongly recommended that organizations not include any content in the body of an e-mail message. If content is included in the body of an e-mail message, the receiving organization is not required to read or respond to it; however, content in an e-mail message should not result in a failed transmission.

File attachment

A file is attached to an e-mail message using the MIME standard with BASE64 encoding. The MIME standard is defined by RFCs 2045, 2046, 2047, and 2049. Only one file attachment may be linked to a single e-mail message. While many e-mail systems provide the capability to link multiple attachments to a single e-mail message, such a configuration creates added complexity and increases the risk of file transport failure.

NOTE

Some e-mail systems automatically create extra MIME attachments. Receiving organizations that have automated e-mail systems must anticipate this possibility and be able to determine which attachment is the actual file.

File attachment size

It is mandatory that the encrypted file attachment must not exceed 1 MB.

The attachment size limit was established to provide consistency in transferring files between trading partners. Many networks and e-mail servers are capable of transporting messages containing much larger file attachments; however, some e-mail servers have attachment size limits that will result in failed transmissions if messages containing larger file attachments are transmitted. Trading partners may agree to support file attachments that exceed the size limitations previously specified.

Naming a file attachment

There are five guidelines to consider regarding naming of a file attachment:

- Sending organizations are not required to follow any specific file naming conventions beyond those requirements stated here when naming data or key files.
- Receiving organizations are not required to use the sender-provided filenames when saving data or key files.

- Receiving organizations must be prepared to rename files as necessary to accommodate their own system requirements/limitations.
- The maximum acceptable length of an attached filename when using SMTP/POP3 transport protocol is 128 characters.
- Allowable characters for attachment filenames are (A-Z) uppercase, 0-9, "_", "-". No other characters will be allowed and the first character must be either (A-Z) uppercase or (0-9).

E-mail message acknowledgment (Evaluate for removal)

E-mail message acknowledgments are optional. Sending organizations may instruct receiving organizations to provide acknowledgment responses (ACKs) to file transmissions. When a sending organization requests an ACK, the receiving organization returns the e-mail subject line from the original transmission to the sending organization with the first three characters changed to ACK. ACKs should only be requested and responded to by automated systems. An organization may NOT request an ACK unless it is also capable of responding to ACK requests by other organizations.

Sending organizations request an ACK by adding a custom SMTP header to the CommonLine, CAM, or CR:C message being sent. This custom header is added to the main SMTP header and implicitly requests the ACK. In addition, this custom header provides an address to which the e-mail message must be sent.

A section of a typical MIME header that requests an ACK might resemble the following:

```
MIME-Version:   1.0
Content-Type:   text/plain; charset=US-ASCII
From:           John Smith <JSmith@isp.com>
To:             Jim Brown <jbrown@isp2.com>
Subject:        COM05 RESP [01] <DOE.825592:ABC95K36>
Date:           Mon, 6 Dec 2001 16:30:00 –0600
X-CL-Ack-To:    <Jack Jones@isp3.com>
```

For acknowledgments, the X-CL-Ack-To: line is the only new portion of the e-mail and will be ignored by any receiving organization that does not support CommonLine, CAM, or CR:C ACK requests.

After receiving an ACK request e-mail, the receiving organization (if it supports ACK functionality) must immediately send an empty message to the e-mail address specified in the custom header. The empty message must echo the subject line of the original message, but change the first three characters from COM or CAM to ACK. An ACK must never request another ACK. In addition, ACK responses must be sent before decryption is attempted. Decryption is considered an application level function rather than transport level (i.e., if a decryption error occurs, the application should prompt the user to contact the sending organization for manual resolution).

To make full use of ACK functionality, a profile entry is needed for each trading partner. This profile will indicate whether an organization supports ACK response functionality. Regardless of whether the receiving organization supports ACK functionality, it is always valid to request an ACK, since such a request will be ignored by organizations that do not support ACKs. It is the sending organization's responsibility to determine if late ACKs should prompt a warning message to the receiving organization, or perhaps necessitate retransmission of the file.

It is the requesting organization's responsibility to determine how ACK profile information is maintained. Maintaining an ACK profile can be a manual or automated process. Automated processes will update the ACK profile whenever a trading partner either requests an ACK or eventually responds to an ACK request

Message acknowledgement timeout

A second optional custom MIME header (X-CL-ACK-Timeout) has also been added to allow trading partners to electronically inform each other of their typical ACK response times. This header is included in the ACK and can be used to set ACK profile information for a timeout parameter on future ACK responses. The timeout parameter indicates how many minutes to wait for an ACK before flagging the transmission as a possible problem.

The X-CL-ACK-Timeout: line must include the time it takes for the receiving organization's e-mail system to deliver the message to the receiving organization's production system, create the ACK message and have the e-mail system send the message back out. The value contained in the X-CL-ACK-Timeout: line cannot account for the time it takes for the ACK message to be brought to the sending organization's e-mail system and delivered to the sending organization's production system. This time must be estimated by the receiving organization and will be added separately.

The first portion of a typical ACK header that supports timeout functionality might resemble the following:

```
MIME-Version:       1.0
Content-Type:       text/plain; charset=US-ASCII
From:               Jim Brown <jbrown@isp2.com>
To:                 jack Jones@isp3.com
Subject:            ACK04 RESP [02] <DOE.999:ABC95K>
Date:               Mon, 6 Dec 2001 16:30:00 –0600
X-CL-Ack-Timeout:   10
```

The timeout number specifies the number of minutes in which an organization expects to be able to respond to an ACK request and can be set large enough to accommodate multiple days. Passing a value of **0** indicates that an organization desires to remove the expected timeout profile from its trading partner's system. If an organization's system supports the timeout feature, it should only watch for this parameter on ACK messages.

THIS PAGE IS INTENTIONALLY BLANK

TRANSPORT PROTOCOLS: File Transfer (FTP)

Introduction

The purpose of this section is to provide CommonLine, CAM, and CR:C participants the procedures necessary for using an FTP system. The FTP protocol is defined in RFC959.

FTP is optimal and recommended for transmitting large files. Differences in an organizations' technology infrastructure creates the need to allow different hardware/software implementations of FTP. The two models of transporting data described in this section are "Push-Push" and "Push-Pull". In both implementations the sending organization (Trading Partner A) begins the transmission process to get data processed by "pushing" the file to the recipient's FTP Server (Trading Partner B). The difference in these implementations is how Trading Partner A receives the response. In the "Push-Push" model, Trading Partner A must be running an FTP server as well so Trading Partner B can "Push" the response back (Diagram 1). In the Push-Pull model, Trading Partner A only runs a FTP client and Trading Partner B makes the processed data available on their server so the Trading Partner A can "pull" the data (Diagram 2).

NOTE

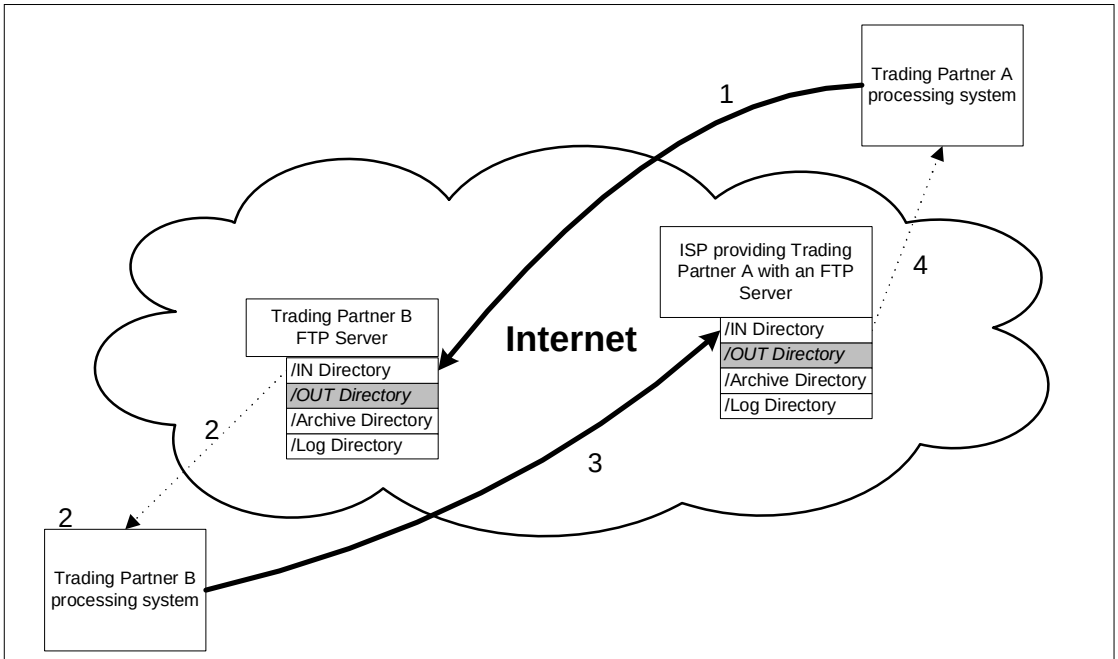
The "Push-Pull" model allows for Trading Partner A to "poll" Trading Partner B's FTP server for unsolicited "responses". FTP clients should not "poll" for data more often than an agreed upon interval set by the trading partner hosting the server. Excessive "polling" by an FTP client could overload FTP server resources and interrupt business for the institution hosting the server (Trading Partner B).

FTP Server implementations must support both passive and active data connections for FTP as defined in RFC-959. All files must be sent in **binary mode**.

NOTE

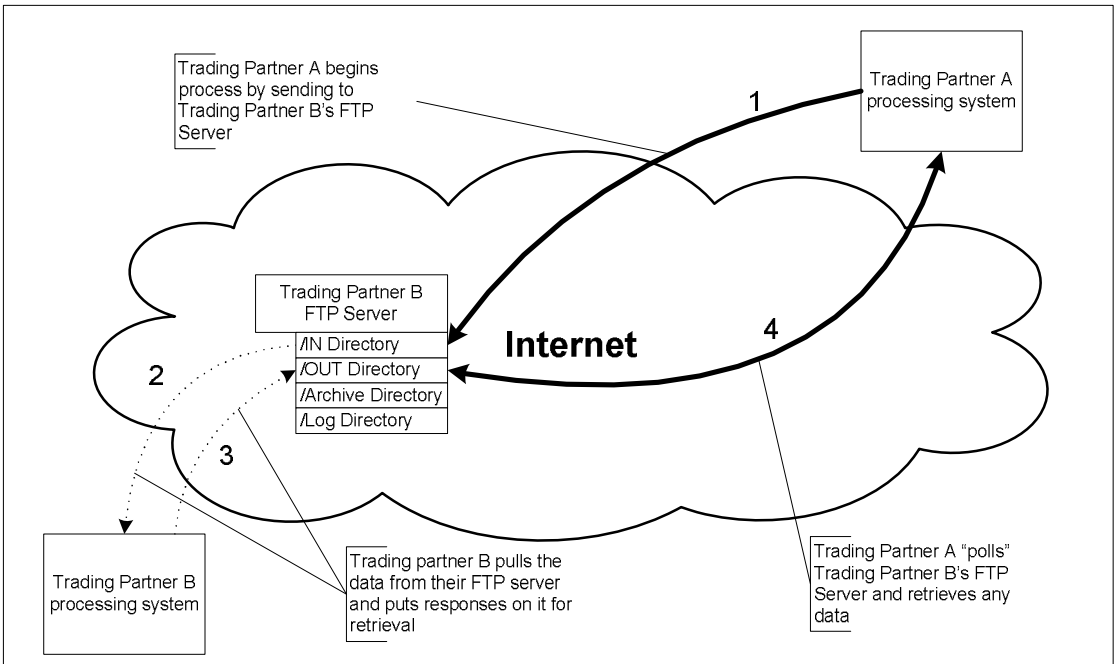
The FTP protocol suffers from an inherent lack of security. While the data being transmitted is secured via PGP encryption prior to transmission, the FTP connection initiation and login occurs using unencrypted data exchanges. Organizations wishing to avoid this weakness may consider replacing FTP with DTS as the transport method. Organizations not ready to transition to DTS should consider adopting secure FTP in the interim. **Therefore it is strongly recommended that existing transport partnerships using traditional FTP be upgraded to secure FTP.** New trading partnerships should consider DTS if possible. Otherwise a secure form of FTP should be used. See the section, "FTP Security" below for more detail.

Diagram 1- PUSH-PUSH Implementation



Trading Partner A has obtained access to an FTP server via an ISP.
Trading Partner B has established an FTP server on-site.
Note: The /OUT directory pictured would be unused in a push-push exchange

Diagram 2 - PUSH-PULL Implementation



Trading Partner A has obtained the necessary client software for FTP transmissions.
Trading Partner B has setup an FTP server on-site, and supports Push-Pull.

FTP business partner responsibilities

For the "Push-Push" implementation each trading partner must obtain access to an FTP server, both trading partners act as clients and hosts. An organization can accomplish this by establishing a full-time FTP service (24 hours per day, 7 days per week) through a local or national ISP. CommonLine, CAM, or CR:C trading partners who have a full time connection to the Internet may choose to install their own FTP servers. Changes necessary for providing access to an organization's system must be in place before its trading partners can begin using FTP to transmit files.

In "PUSH-PULL" implementations only the hosting trading partners must set-up a full-time FTP service (24 hours per day, 7 days per week). The trading partner that is sending and then retrieving the files (Client Trading Partner) only needs FTP client software. Trading partners that are acting as hosts can implement both "Push-Push" and "Push-Pull" architectures utilizing the same FTP Server (see ***FTP Directory Structure*** for implementation differences).

All trading partners must generate public/private key pairs and exchange public keys with their trading partners. Public and private keys are used to encrypt, decrypt, digitally sign, and authenticate data files transmitted via FTP. (See the Security and Authentication: Mechanics and Security and Authentication: Key Management sections for a detailed explanation of these encryption and public/private keys.)

Follow these steps to support to the FTP transport protocol:

1. **[Both Implementations] [Host and Client]**
   Establish the public and a private encryption key pair, using the appropriate encryption standards and user ID. See Key Identifier Specifics section.

2. **[Both Implementations] [Host]**
   Establish an FTP server on-site or obtain access to an FTP server through an ISP.

   - Establish the required directory structure for file handling.
   - Establish a process for generating FTP user IDs for trading partners.

3. **[Both Implementations] [Client]**
   Obtain FTP client software capable of communication with the hosts' FTP server and compliant directory structures.

4. **[Both Implementations] [Host and Client]**
   Create a process for exchanging public keys with CommonLine, CAM, and CR:C trading partners that will do the following:

   - Send the public key to trading partners.
   - Record the trading partners' public keys in a database of public keys (a key ring).
   - Verify public key updates with the sending organization.

- See Key exchange section.

5. **[Both Implementations] [Host and Client]**
   Establish a database of public keys (a key ring) for your trading partners. See Database of Public Keys / Key Ring section.

6. **[Both Implementations] [Host]**
   Create processes to handle incoming data files that will do the following:

   - Acknowledge the incoming files by moving those files to the /ARCHIVE folder.
   - Copy the data files to internal systems for decryption, signature verification and processing.

7. **[Only Push-Pull implementation] [Client]**
   Create process to retrieve ("pull") data files that will do the following:

   - "Poll" for a file in compliant directory structure
   - "Get" files found in /OUT directory
   - Acknowledge receipt of the pulled file by deleting it from the /OUT directory (optionally by moving it to the /ARCHIVE folder if agreed upon by both trading partners)
   - Move the retrieved data files to internal systems for decryption, signature verification, and processing.

8. **[Both implementations] [Client]**
   Create processes to handle outgoing data files that will do the following:

   - Encrypt (lock) the outgoing data files with the receiving organization's public key.
   - Add a digital signature using the sending organization's appropriate private key.
   - Place the data file in the /IN directory on the receiving organization's FTP server.
   - See the Security and Authentication: Mechanics section.

9. **[Both implementations] [Client]**
   Sending organizations should retain a local copy of all transmitted data files in case retransmissions are necessary. Receiving organizations should log incoming data files to permit better tracking of data file exchanges.

FTP server requirements

Unless otherwise agreed to between trading partners, FTP servers must be available full-time (24 hours per day, 7 days per week) as opposed to intermittent or on-demand ISP connections. Trading partners should negotiate service-level agreements involving system availability, capacity, throughput, and/or response time. Organizations should notify their trading partners in the event of unexpected extended FTP server outages. An outage is deemed extended if it lasts more than 2 hours in any 24-hour period.

FTP security

Access to an FTP server must be secured with a password and login ID unique to each organization's application that uses it. The only characters allowed in passwords are A-Z, a-z, 0-9, "_" and "–". After logging in, each organization must be allowed to view only its data area. No special FTP *account* parameter may be used as many FTP servers and clients do not support it properly.

As noted previously, the log-in steps used for traditional FTP are not encrypted meaning that the login name and password could potentially be exposed to anyone monitoring network traffic. While the data itself is adequately protected via PGP, the session itself is exposed and could become compromised. A compromised FTP account means that a malicious third party could log in using "collected" credentials and delete or insert files.

There are secure forms of FTP available which prevent this. These standards include FTP using SSL and an emulated FTP using SSH. It is strongly recommended that trading partners not ready or unable to adopt DTS as the transport method switch all traditional FTP to secure FTP.

Trading partners that host FTP servers should be able to host FTP using both SSH and SSL. If this is not possible, host organizations should choose a protocol that is most compatible with their trading partners.

FTP directory structure

The FTP directory structure described here would be created by the institution that is hosting the FTP server. All directory names must be uppercase and must use the forward slash character (/) instead of the backslash (\ ). Each organization's account that accesses an FTP server must be placed in the organization's HOME (/) directory. If an organization chooses to separate file type families or test and production files on their FTP server(s), this must be done by using either separate IP addresses or separate logons with a completely different HOME (/) directory and **not** with additional directories. This requires FTP clients to have the ability to store separate profile information for CommonLine, CAM, and CR:C destinations at the same organization, which may differ in host name, user name, and/or password. The storage allotted for path information in profiles for trading partners can be no less than 255 characters, including the filenames. Following are the names, descriptions, and assigned privileges, of the subdirectories that should be listed immediately subordinate to the HOME (/) directory:

NOTE

The assigned privileges described in the directory descriptions that follow are the only allowed privileges. Any additional privileges can compromise the security of the network and the receiving organization's system.

Directory name: /IN

Description: This is the directory into which the sending organization (client) places files to be retrieved by the receiving organization (server). Both data files and the key update request file (i.e., CL_COMM_UPDATE message type) are placed here. If a duplicate filename is received, it should trigger a call to the sending organization to notify them of the problem and to determine what should be done with the file. It is the receiving organization's choice whether or not to attempt immediate processing of the file in spite of the error.

Requirement Statement: **This directory is required for both implementations.**

Assigned privileges: Ability to view the list of filenames in the directory.
Ability to view file contents.
Ability to create a file in the directory.

Directory name: /OUT

Description: This is the directory into which the sending organization (server) places files to be retrieved by the receiving organization (client). Both outbound data files and the outbound key update request file (i.e., CL_COMM_UPDATE message type) must be placed here.

Requirement Statement: **This directory is required for the host when supporting the "PUSH-PULL" implementation.**

Assigned privileges: Ability to view the list of filenames in the directory.
Ability to retrieve files.
Ability to delete a file in the directory.

Directory name: /ARCHIVE

Description: This is the directory in which files from the /IN directory are placed by the receiving organization (server) after the files have been promoted to the next step in the receiving organization's process. For most organizations, this means files will be promoted to an internal FTP server and not actually decrypted or processed at the application level until later. The fact that a file has moved from /IN to /ARCHIVE cannot be interpreted to mean the file has been processed at the application level; however it can be

interpreted to mean the file has been received. The filename must be maintained when moving from the /IN directory to the /ARCHIVE directory.

Requirement Statement: **This directory is required for the "PUSH-PUSH" implementation.**

It is optionally used per trading partner agreement in the "PUSH-PULL" implementation. The movement of the file from /OUT to /ARCHIVE can be interpreted that the file was retrieved by the client institution rather than the delete operation serving as the receipt.

Assigned privileges: Ability to view the list of filenames in the directory.
Ability to view file contents.

Directory name: /LOG

Description: This is the directory used to track files placed in the /IN directory by the sending organization (client). If implemented, this directory must contain the log file maintained by the receiving organization (server). The log file must contain the file name, the date and time the file was received, and the date and time the file was processed.

Requirement Statement: **This directory is optional in both implementations.**

Assigned privileges: Ability to view the list of filenames in the directory.
Ability to view file contents.

Due to considerable differences in the way various operating systems track and report paths, it is not reasonable to expect an FTP client to track its position as it navigates between the /IN and /ARCHIVE directories within an organization's directory tree. For this reason, organizations should not change directories once logged into the HOME (/) directory, but rather execute ftp commands including the relative path (required above).

For instance, "put in/COM05_APP_SEND-02-DOE_00999900_EAST-020430999" and "ls in/*"

FTP filename components

The maximum acceptable length of a filename when using the FTP transport protocol is 65 characters. Unlike filenames on SMTP/POP3 attachments the FTP filename conveys the same information as the email subject line which constrains its length to the 65 characters. Allowable characters in filenames are A-Z (uppercase), 0-9, "_", and "–". Unlike the convention used to create e-mail subject lines in the SMTP/POP3 transport protocol, FTP file naming conventions use delimiters (dashes and underscores) between filename components.

Filename components are as follows:

A-B-X_Y_Z-M, where:

A	=	The file type.
B	=	The encryption protocol code.
X_Y_Z	=	The key identifier.
M	=	The unique message identifier.

File types (A)

The first component of the FTP filename is the file type (component A in FTP filename components). The file type identifies the specific type of file being sent. The file type must always occur as the first component of the FTP filename to enable the receiving organization to sort by file type. For a list of valid values for this component of the FTP filename see File Type Component and Payload Type Header table in the Valid Values for Components and Examples section.

Encryption protocol code (B)

The second component of the FTP filename is the encryption protocol code (component B in FTP filename components). The encryption protocol code identifies the type of key and algorithms used to encrypt and digitally sign the message. The 2-digit encryption protocol code is mandatory and must be delimited by dashes (-). See Encryption Protocol Code in Valid Values for Components section. For a list of valid values for this component of the FTP filename see the Encryption Protocol Code table in the Valid Values for Components and Examples section.

Key identifier (X_Y_Z)

The third component of the FTP filename is the key identifier (component X_Y_Z) in the FTP filename components section). The key identifier is subdivided into three segments that identify the sending organization. It is mandatory to separate the file type and encryption protocol code from the key identifier with a dash (–) in the FTP filename. The key identifier in the FTP filename must exactly match the key identifier values contained in the user ID of the public key regardless of the contents of the file being transmitted. See Key Identifier Specifics in the Encryption section for more detailed information on the values of the X_Y_Z components of the Filename.

Unique message identifier (M)

The fourth component of the FTP filename is the unique message identifier (component M in FTP filename components). The unique message identifier is intended to enable trading partners to uniquely identify the message to facilitate file tracking. Only uppercase alphanumeric characters (A-Z and 0-9) and the underscore (_) may be used in the unique message identifier. This message identifier must be unique for at least 90 days.

The unique message identifier is preceded by a dash (-) and is composed as follows:

M where:

M = The unique message identifier. This component of the FTP filename is required.

NOTE

It is strongly recommended that this value be derived from a date/time stamp in the format of "CCYYMMDDHHMMSSnnnnnnnn". No time-zone identifier is included in this scheme. It is assumed that the time stamp is relative to the server time-zone that created the data transmission. Utilizing this format for the "M" section will simplify the compliance with the "90 day" uniqueness requirement, actually making it unique forever.

Refer to the Examples - FTP Filename section to be shown how all these elements are put together.

FTP file sizes

There is no specific size limitation for files transported via FTP; however, practical limits do exist depending on the speed of an organization's Internet connection. Organizations planning to send large files (i.e., **10 MB or larger**) should test such transmissions with their trading partners. If a specific file size proves too large, the file must be split into smaller files with each file's size less than or equal to the size limit, with the resulting file fragments able to stand as complete data files on their own.

NOTE

Files that are formatted in XML may be over 11 times larger than a corresponding flat file record. While encryption implies compression of the documents, the larger original size must be considered when creating the source-unencrypted document. This necessitates a smaller source record count when creating the source documents when compared to a flat file.

Archiving FTP files

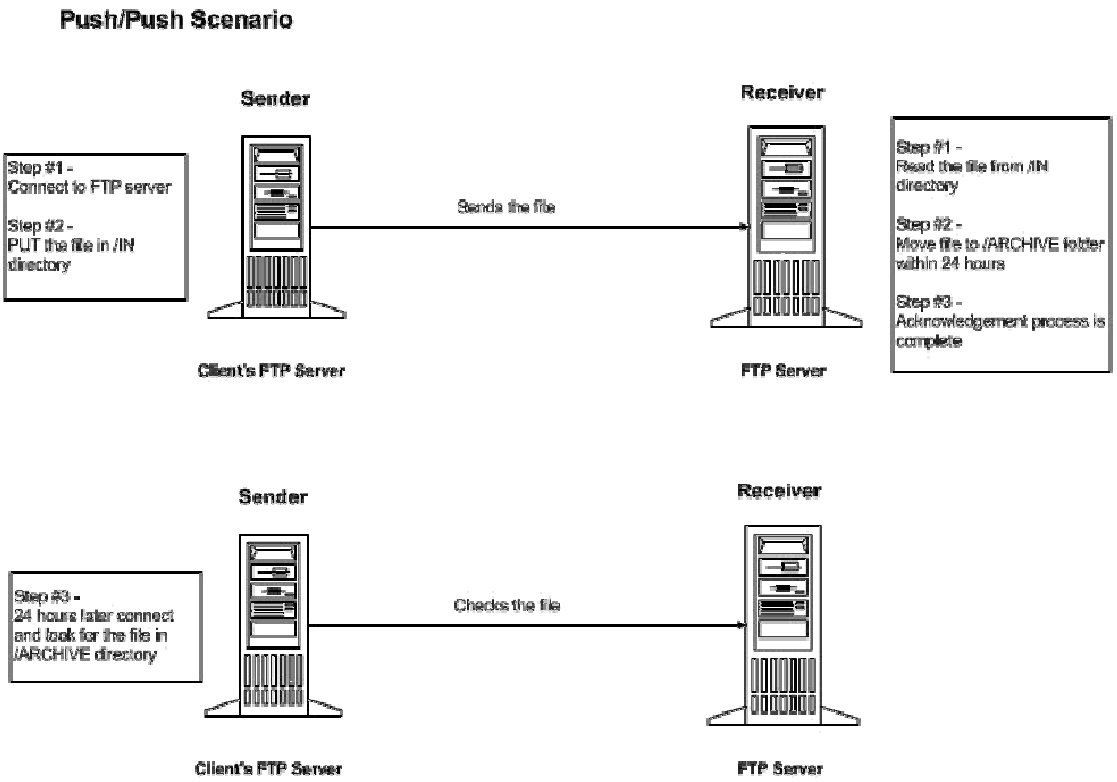
Files must not be deleted from the /IN directory prior to being copied into the /ARCHIVE directory. In addition, files must be promoted out of the /IN directory by the receiving organization (and then moved to the /ARCHIVE directory) within 24 hours of their arrival. Complete files must not remain in the /IN directory for more than 24 hours. If a duplicate filename is encountered in the /ARCHIVE directory, it should trigger a call to the sending organization to notify them of the problem and to determine what should be done with the file.

Organizations must be able to detect that an incoming transfer is complete prior to moving, transferring, or processing a file. The optimal way to accomplish this depends on the server involved. For UNIX systems, ProFTPD can place incoming files into a hidden temporary directory until the transfer is complete—reducing the risk of moving or processing a file while the initial transfer is still in progress. Comparable measures should be considered for other FTP server software.

Acknowledging FTP files

After the sender puts the file in the /IN directory, it is the receiving organization's responsibility to move the file into the /ARCHIVE directory within 24 hours. Once the file has been moved to the /ARCHIVE directory, the receiving organization has thereby acknowledged receipt of the file.

While in most cases, the FTP protocol response codes are adequate for determining a successful transmission, an additional acknowledgement scheme may be utilized. The sending organization may confirm the receiving organization's receipt of the file by checking the /ARCHIVE directory. If the receiving organization does not acknowledge the file within 24 hours, it is the sending organization's responsibility to contact the receiving organization. The sending organization may check earlier than the 24 hours limit and as many times as desired, but the receiving organization need not acknowledge earlier than 24 hours after file delivery.



Purging archived FTP files

After being moved from the /IN directory to the /ARCHIVE directory, files must remain in the /ARCHIVE directory for at least 72 hours after being initially placed into the /ARCHIVE directory. Files in the /ARCHIVE directory must be purged within 90 days from the original transmission date to ensure uniqueness of filenames.

TRANSPORT PROTOCOLS: Data Transport Standard

Introduction

The purpose of this section is to provide CommonLine, CAM, and CR:C participants the procedures necessary for using the DTS Protocol. This section and other references to implementing DTS in this document are in addition to the PESC DTSv1 Specification. All implementations must meet the requirements of the PESC DTSv1 Specification as well as meet the requirements described in this manual.

As with FTP there is more than one implementation model available for DTS. An organizations' technology infrastructure will be the deciding factor for which will be supported. There are three recognized implementations for DTS; "Immediate", "Push-Push", and "Push-Pull".

In the "Immediate" model, the sender of the data receives an immediate response in which the response payload is the end result of the process requested. There is no need for the sender to wait and poll for the response nor is there the need for the recipient to send the response in a separate transaction. The implementation of this model has more to do with the resources of the receiving organization and its ability to process the request within a single https request session than the transport of the data itself. (Diagram 1)

In the "Push-Push" model, Trading Partner A begins the transaction by making a request to the DTS Service provided by Trading Partner B. Trading Partner B performs the tasks necessary to guaranty "processing" of the data sent and then responds to the request acknowledging successful receipt of the data ("…Tran Ack" Payload Type). Once processing of data is completed, Trading Partner B returns the results by making a request to the DTS Service provided by Trading Partner A. Trading Partner A performs the tasks necessary to guaranty "processing" of the data sent and then responds to the request acknowledging receipt of the data. (Diagram 2)

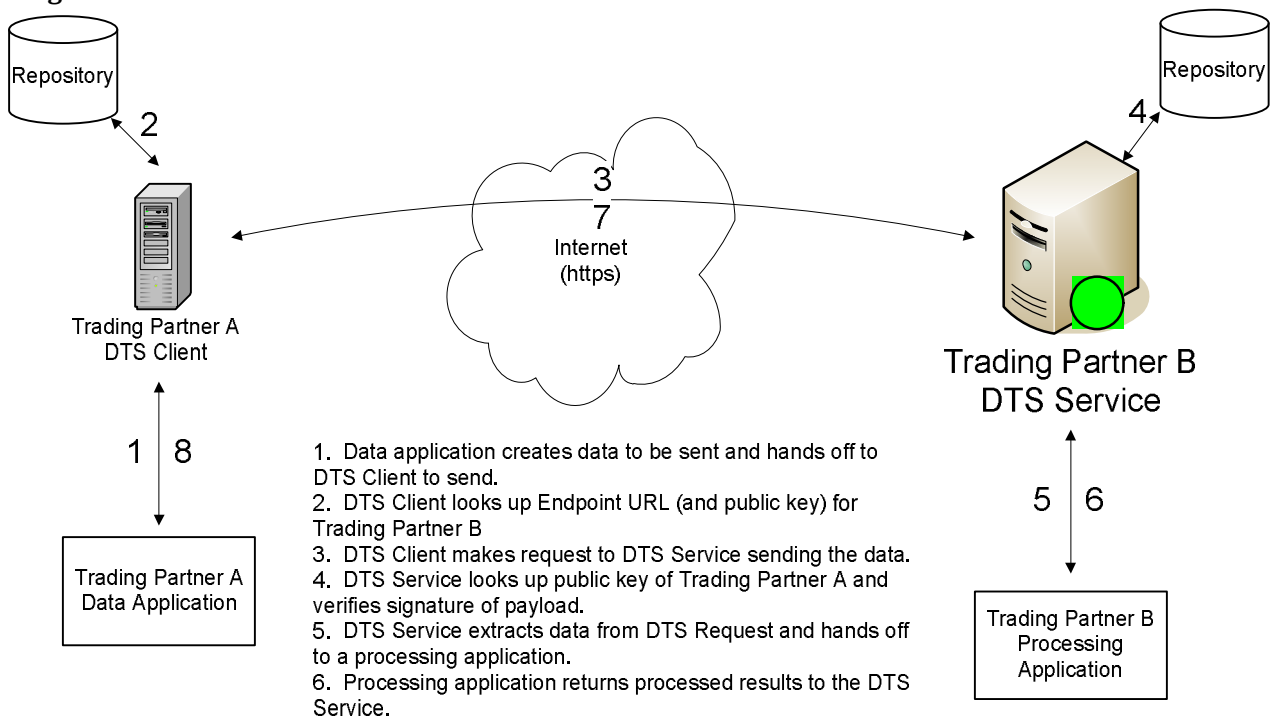
In the "Push-Pull" model, Trading Partner A begins the transaction by making a request to the DTS Service provided by Trading Partner B. Trading Partner B performs the tasks necessary to guaranty "processing" of the data sent and then responds to the request acknowledging successful receipt of the data. Once processing of data is completed, Trading Partner B retains the results of processing until Trading Partner A requests the results. Trading Partner A acquires the results by making another request to the DTS Service provided by Trading Partner B with the DTSRequestPayloadType of "DTS Retrieve" (see Payload Type Header in Valid Values section). Trading Partner B responds to the request with the processed results and removes this data from its processed results cache. To receive all data Trading Partner B may have, Trading Partner A continues to make this request until Trading Partner B returns a DTSResponsePayloadType of "Nothing To Provide". (Diagram 3)

Note

DTS Clients "polling" for data should not do so more often than an agreed upon interval. Excessive "polling" by the client could overload Web Server resources and interrupt business for the institution providing the service (Trading Partner B).

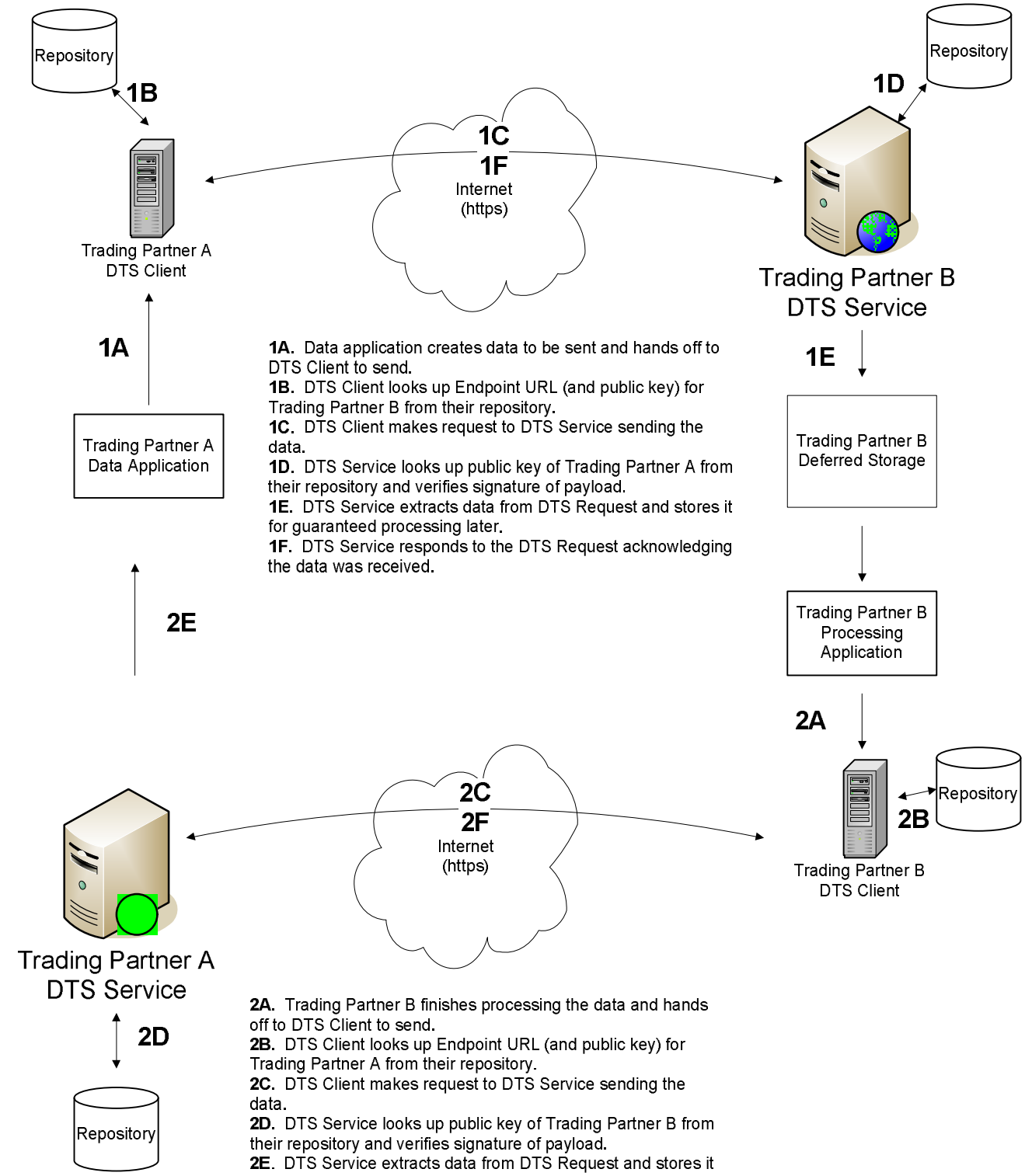
DTS Services supporting Push-Pull must allow for sufficient storage to cache results not yet collected by their trading partners to prevent service interruptions.

Diagram 1 – "Immediate"



1. Data application creates data to be sent and hands off to DTS Client to send.
2. DTS Client looks up Endpoint URL (and public key) for Trading Partner B
3. DTS Client makes request to DTS Service sending the data.
4. DTS Service looks up public key of Trading Partner A and verifies signature of payload.
5. DTS Service extracts data from DTS Request and hands off to a processing application.
6. Processing application returns processed results to the DTS Service.
7. DTS Service responds to the DTS Request with the processed results.
8. DTS Client uses public key (step 2) to verify signature and then hands processed results back to the data application.

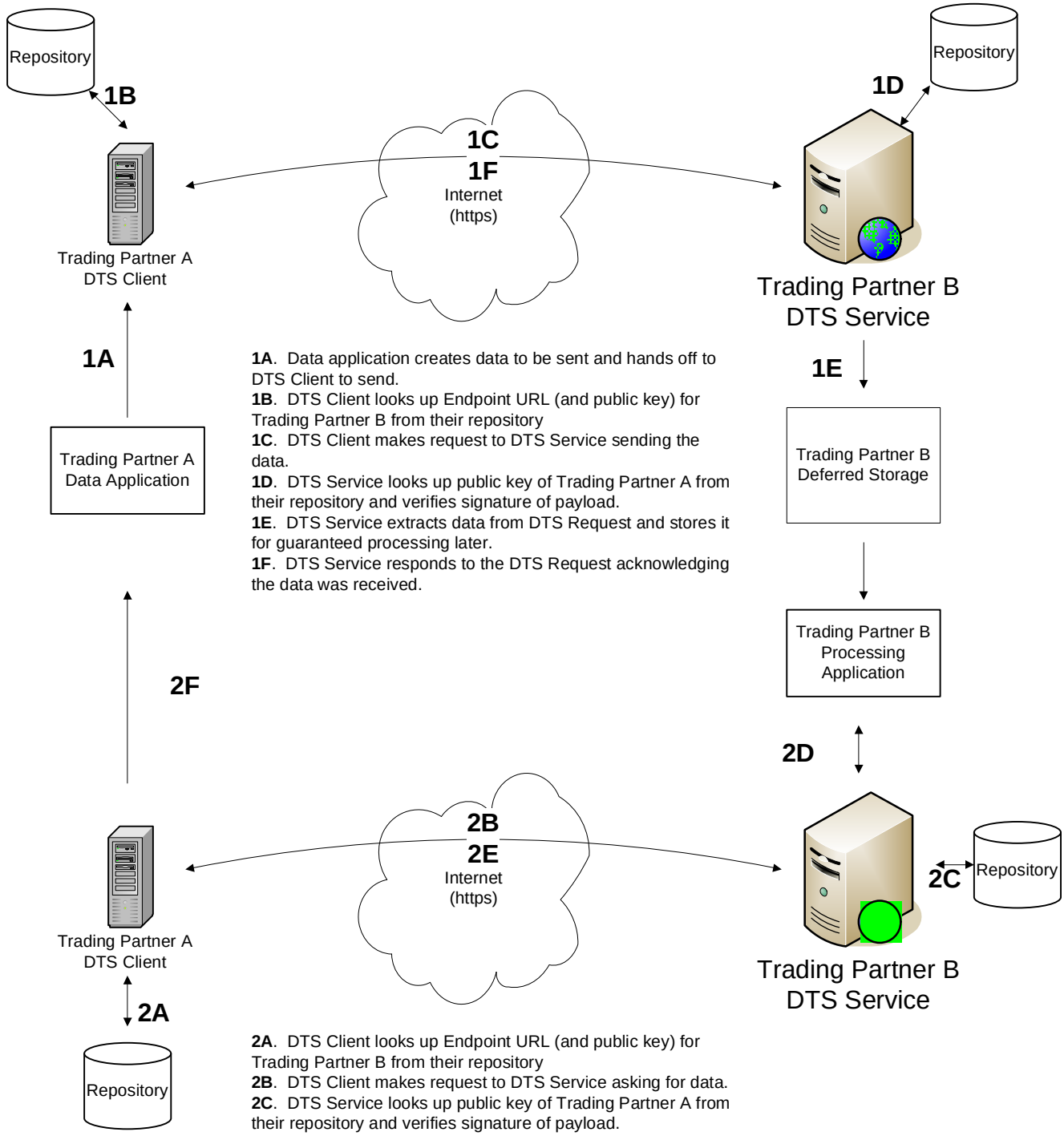
Diagram 2 – "Push-Push"



1A. Data application creates data to be sent and hands off to DTS Client to send.
1B. DTS Client looks up Endpoint URL (and public key) for Trading Partner B from their repository.
1C. DTS Client makes request to DTS Service sending the data.
1D. DTS Service looks up public key of Trading Partner A from their repository and verifies signature of payload.
1E. DTS Service extracts data from DTS Request and stores it for guaranteed processing later.
1F. DTS Service responds to the DTS Request acknowledging the data was received.

2A. Trading Partner B finishes processing the data and hands off to DTS Client to send.
2B. DTS Client looks up Endpoint URL (and public key) for Trading Partner A from their repository.
2C. DTS Client makes request to DTS Service sending the data.
2D. DTS Service looks up public key of Trading Partner B from their repository and verifies signature of payload.
2E. DTS Service extracts data from DTS Request and stores it to process later.
2F. DTS Service responds to the DTS Request acknowledging the data was received.

Diagram 3 – "Push-Pull"



1A. Data application creates data to be sent and hands off to DTS Client to send.
1B. DTS Client looks up Endpoint URL (and public key) for Trading Partner B from their repository
1C. DTS Client makes request to DTS Service sending the data.
1D. DTS Service looks up public key of Trading Partner A from their repository and verifies signature of payload.
1E. DTS Service extracts data from DTS Request and stores it for guaranteed processing later.
1F. DTS Service responds to the DTS Request acknowledging the data was received.

2A. DTS Client looks up Endpoint URL (and public key) for Trading Partner B from their repository
2B. DTS Client makes request to DTS Service asking for data.
2C. DTS Service looks up public key of Trading Partner A from their repository and verifies signature of payload.
2D. DTS Service looks for processed data for Trading Partner A.
2E. DTS Service responds to the DTS Request with any data available, if no data is available the returns "Nothing to Provide".
2F. DTS Client verifies signature from Trading Partner B, and extracts payload for processing.

DTS Web Server Requirements

Unless otherwise agreed to between trading partners, DTS Web servers must be available full-time (as close to 24 hours per day/7 days per week as possible acknowledging maintenance periods) as opposed to intermittently or using an on-demand ISP connection. A DTS Web server must be running SSL to ensure the data stream is encrypted via HTTPS. Trading partners should negotiate service-level agreements involving system availability, capacity, throughput, and/or response time. Organizations should notify their trading partners in the event of unexpected extended Web server outages. An outage is deemed extended if it lasts more than 2 hours in any 24-hour period.

Organizations implementing a DTS Service, must obtain an X509 certificate of a type suitable for the site's web server. DTS does not mandate a particular web server (e.g. Microsoft IIS or Apache). Each organization should consult its web server's documentation for details on obtaining and installing an X509 certificate to enable the web server to support the HTTPS protocol. It is not necessary for organizations to exchange public keys used by web servers to enable HTTPS.

DTS Security and Authentication

DTS implementations are secure via the use of SSL encryption of the HTTP stream (HTTPS). The authenticity of the message received is validated via Digital Signatures implemented through the use of X.509 Certificates that have been originally obtained from a trusted Certificate Authority and subsequently verified as valid by the receiving organization.

The use and exact implementation of Digital Signatures with DTS is specified in the PESC DTS v1.0.1 Specification.

For more information concerning Security and Authentication via SSL and X.509 certificates see the Security and Authentication: Mechanics and Security and Authentication: Key Management sections.

Required Encryption Keys

Implementing SSL for encryption eliminates the need for encryption key exchange. The encryption and all necessary steps for implementing it are already handled under the HTTPS communication. See the SSL Encryption section for more information.

Supporting Digital Signatures

While the PESC DTS v1.0.1 Specification requires the use of digital signatures, it does not explicitly state how to do so. The following are additional requirements which are necessary to support the DTSv1.0.1 requirements for using digital signatures:

- Obtain a valid X.509 Certificate (see Key Generation - X.509 Certificates)

- Exchange the X.509 Certificate being used for Digital Signatures (see Key Exchange - X.509 Certificate)
- Implement some type of Database for public keys (see Database of Public Keys / Key Ring)

DTS Payload sizes

DTS can support both small and large payloads with maximum benefit coming from smaller payloads and immediate processing. While larger payloads can be accommodated, various levels of testing and development have proven that 50 megabytes (before compression) is the threshold for reliable transport. Thus, 50 megabytes (before compression) is the maximum size of a payload to be transported via DTS

DTS End-Points

Participants using DTS need to know where to find their trading partner on the internet. Each service provider needs an address and every participant communicating with that service provider needs to know where to reach the provider, while allowing the provider some flexibility to manage changes to their own configuration (see Database of Public Keys / Key Ring).

For example, if School B wants to publish that their address for DTS has moved to another location within the school, all their trading partners need to get the new location.

All service providers must establish a full internet address for DTS services they provide. This DTS endpoint looks like a web address, but is used differently by the underlying technology. A service provider may have one endpoint for one type of data and another for different type of data, as well as segregating the use of test and production environments. Client organizations should consider this service side flexibility when creating their DTS Client.

Some examples:

https://dts.studentloans.lender.com/axis/services/DTSsubmit
https://dts.TESTstudentloans.lender.com/axis/services/DTSsubmit

https://www.guarantor.org/DTSsend/dts.asmx
https://www.guarantor.org/DTSsend/dtsCRC.asmx

End-point information should be exchanged during the initial negotiations for beginning electronic transmission implementing DTS.

DTS Business Partner Responsibilities

The following outlines the responsibilities and requirements that an organization must perform in order to implement a DTS Client and DTS Service. The list is not a comprehensive list of everything that could be done in establishing a web client or service; consult your organizations technology department for specific guidelines and procedures.

Note:
Additional information for implementing a DTS Client and Service can be obtained from the PESC document

"DTS Reference Implementation Guide". This guide illustrates how to create DTS Specification compliant applications. It does not define values of SOAP elements or dictate communication flow which is defined in this Technical Manual (See DTS SOAP Components).

DTS Client

1. Acquire an X.509 certificate from a Certificate Authority to be used for digitally signing the payload (Refer to the X.509 sections within Security and Authentication: Key Management).
2. Exchange the public certificate with trading partner(s) and gather other pertinent information.
3. Protect the private key associated to certificate.
4. Establish an internal repository to hold trading partners' X.509 certificates, end-points, and capabilities.
5. Obtain or create software that is capable of communicating with DTS Services as defined by the PESC DTS Specification.
6. Ensure connectivity paths through corporate firewalls.
7. Establish processes to handle responses from a DTS Service.
8. If implementing only a DTS Client, establish "polling" mechanism to retrieve deferred processed results from a DTS Service.

DTS Service

1. Acquire an X.509 certificate from Certificate Authority to be used for digitally signing the payload (Refer to the X.509 sections within Security and Authentication: Key Management).
2. Acquire an X.509 certificate from Certificate Authority to be installed on web server to enable encryption via SSL (https). (Refer to the SSL sections within Security and Authentication: Mechanics).
3. Exchange the public certificate used for signing and other pertinent information with trading partner(s).
4. Protect the private key associated to certificates.
5. Establish an internal repository to hold trading partners' X.509 certificates.
6. Obtain or create Web Service software and site that is capable of communicating with DTS Clients as defined by the PESC DTS Specification.
7. Ensure Web Service access through corporate firewalls.
8. Establish processes to handle requests from and responses to a DTS Client.
9. Determine back end processing capabilities in order to answer service expectations appropriately.

Timeout methodology for DTS Client and Service implementation

A DTS Service should interrogate the Type of data, service expectation, and payload size (to get an idea of volume); and perform an assessment based on internal system capabilities to determine if the request can be accommodated within the default HTTPS request time out of 60 seconds. The service should respond with an acknowledgement of receiving the data (once stored on their system) and indicate that it will be processed "Deferred" in the case that this assessment predicts that a timeout would occur. This allows the Client to know the request was received successfully but could not be processed within the allotted time.

A DTS Client should allow the default of 60 seconds per request for some type of response to be returned.

The HTTPS request timeout of 60 seconds has been proven to be sufficient for the successful transmission of data as long as the maximum payload size limit is not exceeded. Actual processing of the data may take more time depending on the type of data, payload size, and the specific transactions involved.

Both of these guidelines can be overlooked and/or ignored with a mutual agreement between two trading partners, though such agreements can not be forced upon a third partner.

DTS SOAP Components

DTS Header elements

The DTS protocol is defined in the PESC DTSv1.0.1 Specification. While the PESC Specification identifies the SOAP elements necessary for data exchange, the actual values contained in those elements were left to be defined by the various industry segments implementing DTS. However, some header elements have such a distinct definition that the DTS Specification defines the allowable values for them as well. This section defines the values for the remaining SOAP elements defined in the PESC Specification. Where the value of the element is explicitly defined in the PESC DTS Specification, it will be noted.

DTSRequestRouting and DTSResponseRouting

This header element is a complex element which contains additional information regarding the routing of the DTS request or response. The child elements are as follows:

Child element	Value Definition
sourceID	Identifies the source of the request and is used by the recipient to find the public key of the sender in their repository. The unique identification number assigned by ED or NCHELP. Made up of the ED – Numeric ID and non-ED Branch ID (if used) in the format of “Y.Z”. Correlates to the Key Identifier described in Key Identifier Specifics. ED - Numeric ID values are assigned by ED (i.e.; 3-digit ID’s to guarantors, 6-digit ID’s to lenders and servicers, and 8-digit ID’s to schools). NCHELP - Alphanumeric "3-8 character" ID values are assigned by NCHELP. Only servicers will be assigned this type of ID number. The length of this non-ED assigned identifier is not related to the type of servicer to which it is assigned.

	This value is required **NOTE** It is understood that the NCHELP identification number will be different than the identification number assigned by ED. Zeros and spaces must not be used as padding. Only zeros that are part of the actual ED ID may be used.
sourceIDSubCode	A constant value of **DOE or VID. This value is required.** This is the "X" component of the key identifier described in Key Identifier Specifics. This value is used in conjunction with the SourceId value in order for the recipient to find the public key of the sender in their repository.
recipientID	Identifies the recipient of the request. The data contained in this element is the same format as the SourceId and would be used by the sender to look up the DTS End-Point for that institution. **This value is required.** A single institution may implement a DTS service for multiple agencies. This value allows routing for appropriate processing.
recipientIDSubCode	A constant value of **DOE or VID. This value is required.** This is the "X" component of the key identifier described in Key Identifier Specifics. This value is used in conjunction with the RecipientId value in order for the sender to find the DTS End-Point for the recipient.
UUID	A universally unique identifier for the synchronous DTS communication. A UUID is essentially a 16-byte number and in its canonical form a UUID may look like this: 550e8400-e29b-11d4-a716-446655440000 **This value is required.** (See http://en.wikipedia.org/wiki/UUID for more information)
transmissionDateTime	The date and time stamp of the request or response in GMT format. **This value is required.** This element is strictly defined by the PESC Specification.

DTSRequestPayloadType and DTSResponsePayloadType

The value of these header elements define the type of data that is contained in the payload (SOAP:Body) of the request or response. The purpose of this element is to be able to identify the payload for routing purposes without being required to open the payload directly. This element corresponds to the File Type component described in the POP/SMTP and FTP sections. For a list of valid values to be placed in this element see File Type Component and Payload Type Header table in the Valid Values for components and Examples section. **These values are required.**

DTSRequestServiceExpectation and DTSResponseAcknowledge

The value of the DTSRequestServiceExpectation element is used by the receiving service to determine how the sender would like the payload to be processed. In the response to the request, the value of DTSResponseAcknowledge is used to indicate to the original sender how the payload was actually processed. **These values are required.**

Element Value	Description/Meaning
Immediate	In the Request, the sender is requesting that the payload be processed immediately and results be returned as the payload of the synchronous response. In the Response, the recipient is informing the sender that the payload was processed and the result is returned as the payload. **See DTS Transaction Flows for scenario.**
Deferred	In the Request, the sender is indicating that the payload may be processed in a batch manner and the sender is not expecting the results immediately. In the Response, the recipient is indicating that the payload will be processed at a later time. The payload in the synchronous response could contain a look up token to be resent by the client at a later time (see Addendum #3: Use of Response "Tokens" in DTS). **See DTS Transaction Flows for scenario.**

DTSRequestSignature and DTSResponseSignature

The values of these elements are fully defined by the PESC Specification and are required.

These elements hold the digital signature of the compressed and encoded payload. The data in these elements must be base64 encoded.

Implementing these elements allow for the assurance that the data sent was indeed sent by the agency listed as the source in the routing elements and also ensures that the data was not modified in transit. (See Digital Signatures section for more information)

DTSRequestPayloadBytes and DTSResponsePayloadBytes

The values of these elements are fully defined by the PESC Specification and are required.

These elements hold the decompressed byte counts of the respective payloads. Every DTS Service (and client) will have system resource constraints (memory, disk space, etc.) that should be taken into consideration when attempting to decompress the payloads. The byte counts are required to provide a reliable method to assist the application in determining the best method to decompress the payloads.

DTS Body Element

DTSRequest and DTSResponse

These elements are the children of the SOAP:Body. The values of these elements are the business transactions to be transported and are considered the payloads of DTS transmissions. The actual contents are defined (described) by the DTSRequestPayloadType and DTSResponsePayloadType header elements.

DTS Error Handling

The PESC DTS v1.0.1 Specification defines the SOAP faults that are to be used when the communication fails due to not complying with the DTS Specification itself. Refer to the specification for explicit details of generating or processing the Faults.

This section will expand on the DTS.Application fault defined in the DTS Specification which is described to be used when a backend processing error occurs when the DTS Service attempts to process the payload (either fully process or place in a queue for processing).

A SOAP fault has three elements that are used within DTS to ensure interoperability; Fault Code, Fault String, and Fault Detail.

When a DTS Service can not continue reliable processing the following will be sent:

Fault Code	`DTS.APPLICATION`
Fault String	`Backend Processing Error`
Fault Detail	*See Below*

The first two elements above allow the DTS Client that receives the fault to immediately know that while the DTS Specification was met and communication to the service was successful some other error occurred that has halted processing.

The third element of the SOAP fault, Fault Detail, is left to the organization providing the service to define and populate just so long as it meets the rest of the fault structure. The text provided in this element can tell of the error as explicitly or generically as desired by the organization.

Note

It is strongly recommended that the Fault Detail contain some type of information that will allow the organization that created it to quickly rectify the problem when contacted by the organization that received it. This could include a full stack trace of the error or a generic description of where it took place or just an identifier of some kind that will provide the starting place for researching more explicit logs at the service.

Organizations that have defined the Fault Detail could also provide those explicit descriptions to the institution creating the client during initial negotiations for implementing DTS along with the actions that should be taken for each.

This fault is used for any error that occurs on the Service system that will cause the data/transaction to not be processed for reasons other than transmission or SOAP. The details of this fault are of little consequence other than communicating to the Client that a fatal error has occurred.

DTS Transaction Flows

This section provides scenarios and expected behavior of DTS Clients and Services for some of the responsibilities listed in the DTS Business Partner Responsibilities and other business scenarios. Where applicable portions of the SOAP elements will be listed as an example of how the communication could behave.

DTS Client

Obtain/create software capable of communicating with a DTS Service

1. Should be able to accept data from another application or create the data to send
2. Should communicate with an internal repository to retrieve information elements necessary for sending to recipient's end point and verifying their signature (see Database of Public Keys / Key Ring)
3. Must create DTS compliant SOAP envelope
4. Invoke a service and handle synchronous response from the service
    a. Scenarios
        i. If "Immediate" processing requested and responded (or "Deferred" requested and "Immediate" response), then pass response data to internal applications for processing

Example of SOAP Header Elements (see PESC Specification for exact examples, this is only provided to show flow of communication for the scenario):

Client:
<DTSRequestPayloadType>**CR:C01 Request**</DTSRequestPayloadType>
<DTSRequestServiceExpectation>**Immediate**</DTSRequestServiceExpectation>
Or
<DTSRequestServiceExpectation>**Deferred**</DTSRequestServiceExpectation>

Payload contains CR:C Request document

Service:
<DTSResponsePayloadType>**CR:C01 Response**</DTSResponsePayloadType>
<DTSResponseAcknowledge>**Immediate**</DTSResponseAcknowledge>
Payload contains CR:C Response document

The DTS Client receiving the above response from the DTS Service must recognize that the service is indicating that the payload is a CR:C01 Response and was processed immediately. Thus, the payload of the DTSResponse is the processed result of what was sent and therefore pull the payload out to be processed by their other systems.

ii. If "Deferred" processing requested and responded (or "Immediate" requested and "Deferred" response), then accept acknowledgement of successful transmission and;

1. **Push-Push implementation**; wait for the response to be returned to the DTS Service.

Example of SOAP Header Elements (see PESC Specification for exact examples, this is provided to show flow of communication for the scenario):

Client:
<DTSRequestPayloadType>**CR:C01 Request**</DTSRequestPayloadType>
<DTSRequestServiceExpectation>**Immediate**</DTSRequestServiceExpectation>
Or
<DTSRequestServiceExpectation>**Deferred**</DTSRequestServiceExpectation>
Payload contains CR:C Request document

Service:
<DTSResponsePayloadType>**CR:C01 Request Tran Ack**</DTSResponsePayloadType>
<DTSResponseAcknowledge>**Deferred**</DTSResponseAcknowledge>
Payload is empty (8 bytes when compressed and encoded in SOAP)
Or
Payload contains services proprietary token (Addendum #3: Use of Response "Tokens" in DTS); Client must not error even if not supporting "tokens".

2. **Push- Pull implementations (**DTS Client Only sites**)** – Create process to retrieve data from trading partners

   a. Invoke DTS Service of trading partner with "DTS Retrieve" payload type.
   b. Extract payload of the DTS Service's response.
   c. Continue invoking service until the response payload type of "Nothing to Provide" is received and payload is empty

Example of SOAP Header Elements (see PESC Specification for exact examples, this is provided to show flow of communication for the scenario):

Client:
<DTSRequestPayloadType>**DTS Retrieve**</DTSRequestPayloadType>
<DTSRequestServiceExpectation>**Immediate**</DTSRequestServiceExpectation>

Payload is empty (8 bytes when compressed and encoded in SOAP)

Service Response 1:
<DTSResponsePayloadType>**CR:C01 Response**</DTSResponsePayloadType>
<DTSResponseAcknowledge>**Immediate**</DTSResponseAcknowledge>
Payload contains CR:C Response document

Service Response 2:
<DTSResponsePayloadType>**Nothing to Provide**</DTSResponsePayloadType>
<DTSResponseAcknowledge>**Immediate**</DTSResponseAcknowledge>
Payload is empty (8 bytes when compressed and encoded in SOAP)

DTS Service

Obtain/create DTS Web Service capable of communicating with a DTS Client

1. Should communicate with repository to retrieve public key of sender and verify the digital signature
    a. Scenarios:
        i. Data received and "Immediate" processing requested
            1. And the service application supports immediate processing of payload (and volume) in synchronous communication;
                a. Process payload
                b. Respond to request with processed result data as payload and indicating "Immediate" processing

Example of SOAP Header Elements (see PESC Specification for exact examples, this is only provided to show flow of communication for the scenario):

Client:
<DTSRequestPayloadType>**CR:C01 Request**</DTSRequestPayloadType>
<DTSRequestServiceExpectation>**Immediate**</DTSRequestServiceExpectation>
Payload contains CR:C Request document

Service:
<DTSResponsePayloadType>**CR:C01 Response**</DTSResponsePayloadType>
<DTSResponseAcknowledge>**Immediate**</DTSResponseAcknowledge>
Payload contains CR:C Response document

            2. And the service application does **not** support immediate processing of payload in synchronous communication;
                a. Save the payload for processing at a later time
                b. Respond to request acknowledging receipt of data and indicating "Deferred" processing
                i. If service is capable of handling tokens; payload could contain the token (Addendum #3: Use of Response "Tokens" in DTS)
                ii. If service does not incorporate tokens, Payload will be empty

Example of SOAP Header Elements (see PESC Specification for exact examples, this is provided to show flow of communication for the scenario):

Client:
<DTSRequestPayloadType>**CR:C01 Request**</DTSRequestPayloadType>

<DTSRequestServiceExpectation>**Immediate**</DTSRequestServiceExpectation>
Payload contains CR:C Request document

Service:
<DTSResponsePayloadType>**CR:C01 Request Tran Ack**</DTSResponsePayloadType>
<DTSResponseAcknowledge>**Deferred**</DTSResponseAcknowledge>
Payload is empty (8 bytes when compressed and encoded in SOAP)
Or
Payload contains services proprietary token (Addendum #3: Use of Response "Tokens" in DTS)

ii. Data received and "Deferred" processing requested;
The service application could support immediate processing of payload in a synchronous communication. In which case the following options could be considered depending on internal practices:

1. Service Application handles immediate processing
    a. Process payload
    b. Respond to request with processed result data as payload and indicating "Immediate" processing

Example of SOAP Header Elements (see PESC Specification for exact examples, this is only provided to show flow of communication for the scenario):

Client:
<DTSRequestPayloadType>**CR:C01 Request**</DTSRequestPayloadType>
<DTSRequestServiceExpectation>**Deferred**</DTSRequestServiceExpectation>
Payload contains CR:C Request document

Service:
<DTSResponsePayloadType>**CR:C01 Response**</DTSResponsePayloadType>
<DTSResponseAcknowledge>**Immediate**</DTSResponseAcknowledge>
Payload contains CR:C Response document

2. Service Application handles immediate processing but clients' request takes precedence
    a. Save the payload for processing at a later time
    b. Respond to the request acknowledging receipt of data and indicating "Deferred" processing.

Example of SOAP Header Elements (see PESC Specification for exact examples, this is provided to show flow of communication for the scenario):

Client:
<DTSRequestPayloadType>**CR:C01 Request**</DTSRequestPayloadType>
<DTSRequestServiceExpectation>**Deferred**</DTSRequestServiceExpectation>
Payload contains CR:C Request document

Service:
<DTSResponsePayloadType>**CR:C01 Request Tran Ack**</DTSResponsePayloadType>
<DTSResponseAcknowledge>**Deferred**</DTSResponseAcknowledge>
Payload is empty (8 bytes when compressed and encoded in SOAP)
Or
Payload contains services proprietary token (Addendum #3: Use of Response "Tokens" in DTS)

iii. "DTS Retrieve" Payload type
   1. Service application should check internal system for data of any type to be given to the organization defined by the SourceID header element
      a. Provide data as Response payload
      b. Respond with "Nothing to Provide"
         i. Respond with empty payload if nothing to provide

Example of SOAP Header Elements (see PESC Specification for exact examples, this is provided to show flow of communication for the scenario):

Client:
<DTSRequestPayloadType>**DTS Retrieve**</DTSRequestPayloadType>
<DTSRequestServiceExpectation>**Immediate**</DTSRequestServiceExpectation>
Payload is empty (8 bytes when compressed and encoded in SOAP)
Or
Payload contains service's proprietary token received in response of previous transmission (Addendum #3: Use of Response "Tokens" in DTS)

Service Response 1:
<DTSResponsePayloadType>**CR:C01 Response**</DTSResponsePayloadType>
<DTSResponseAcknowledge>**Immediate**</DTSResponseAcknowledge>
Payload contains CR:C Response document

Service Response 2:
<DTSResponsePayloadType>**Nothing to Provide**</DTSResponsePayloadType>
<DTSResponseAcknowledge>**Immediate**</DTSResponseAcknowledge>
Payload is empty (8 bytes when compressed and encoded in SOAP)

Service Response 3:
<DTSResponsePayloadType>**Nothing to Provide**</DTSResponsePayloadType>
<DTSResponseAcknowledge>**Deferred**</DTSResponseAcknowledge>
Payload contains service's proprietary token (Addendum #3: Use of Response "Tokens" in DTS)

2. Create the DTS response to the request

a. All requests are required to get a valid DTS response depending on processing of the request
    i. At the minimum the DTSResponsePayloadType “Tran Ack” that correlates to the DTSRequestPayloadType should be sent with the DTSResponseAcknowledge of “Deferred”
    ii. See other examples above
b. Sign the payload with the private key associated to the previously shared X.509 certificate (containing the public key and certificate chain) meant for digital signing. (See Digital Signatures and Key Generation - X.509 Certificates and Key Exchange - X.509 Certificate)

Valid Values for Components and Examples

Valid Values

This section contains the acceptable and valid values to be used for parts of a subject line (POP/SMTP protocol), a filename (FTP protocol), and SOAP:Header elements (DTS protocol). They have been grouped together so that the congruent and consistent nature of these values, their purpose, and usage can be readily evident.

Encryption Protocol Code

The encryption protocol code identifies the type of key and algorithms used to encrypt and digitally sign the data file.

Valid encryption protocol codes are as follows:

Code	Encryption Method
00	No encryption used. This code is provided for testing purposes or internal use only. **Production data must not be exchanged without encryption.**
02	OpenPGP as defined in RFC 2440 (See ***Appendix A: OpenPGP Specifics***.)
03	X.509 Certificate - Only implemented for exchanging of Certificates to be used for DTS Digital Signatures, only valid in CL COMM UPDATE subject/filename or DTS Payload type.

File Type Component and Payload Type Header

The following table lists the File Type component of the subject line (Described in POP/SMTP section), the File Type component of the Filename (described in FTP section), and the Payload Type Header element (described in the DTS section).

Description	Pop3 File Type Component Of Subject Line	FTP File Type Component Of Filename	DTSRequestPayloadType and/or DTSResponsePayloadType SOAP Header Value
Application Send File, CommonLine Release 4	COM04 APP SEND	COM04_APP_SEND	COM04 APP SEND
Application Send File, CommonLine Release 4 received by DTS Service acknowledgement.	N/A	N/A	COM04 APP SEND TRAN ACK (Response only)
Application Send File, CommonLine Release 5	COM05 APP SEND	COM05_APP_SEND	COM05 APP SEND

Description	Pop3 File Type Component Of Subject Line	FTP File Type Component Of Filename	DTSRequestPayloadType and/or DTSResponsePayloadType SOAP Header Value
Application Send File, CommonLine Release 5 received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	COM05 APP SEND TRAN ACK (Response only)
Response File, CommonLine Release 4	COM04 RESP	COM04_RESP	COM04 RESP
Response File, CommonLine Release 4 received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	COM04 RESP TRAN ACK (Response only)
Response File, CommonLine Release 5	COM05 RESP	COM05_RESP	COM05 RESP
Response File, CommonLine Release 5 received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	COM05 RESP TRAN ACK (Response only)
Disbursement Roster File, CommonLine Release 4	COM04 DISB ROST	COM04_DISB_ROST	COM04 DISB ROST
Disbursement Roster File, CommonLine Release 4received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	COM04 DISB ROST TRAN ACK (Response only)
Disbursement Roster File, CommonLine Release 5	COM05 DISB ROST	COM05_DISB_ROST	COM05 DISB ROST
Disbursement Roster File, CommonLine Release 5 received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	COM05 DISB ROST TRAN ACK (Response only)
Disbursement Roster Acknowledgment File, CommonLine Release 4	COM04 DISB RESP	COM04_DISB_RESP	COM04 DISB RESP
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	COM04 DISB RESP TRAN ACK (Response only)
Disbursement Roster Acknowledgment File, CommonLine Release 5	COM05 DISB RESP	COM05_DISB_RESP	COM05 DISB RESP
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	COM05 DISB RESP TRAN ACK (Response only)
Disbursement Forecast File, CommonLine Release 5	COM05 DISB FORC	COM05_DISB_FORC	COM05 DISB FORC

Description	Pop3 File Type Component Of Subject Line	FTP File Type Component Of Filename	DTSRequestPayloadType and/or DTSResponsePayloadType SOAP Header Value
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	COM05 DISB FORC TRAN ACK (Response only)
Change Transaction Send File, CommonLine Release 4	COM04 CHNG SEND	COM04_CHNG_SEND	COM04 CHNG SEND
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	COM04 CHNG SEND TRAN ACK (Response only)
Change Transaction Send File, CommonLine Release 5	COM05 CHNG SEND	COM05_CHNG_SEND	COM05 CHNG SEND
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	COM05 CHNG SEND TRAN ACK (Response only)
CAM submission file	CAMS01	CAMS01	CAMS01
CAM submission file received by DTS Service acknowledgement. Payload is empty. This payload type replaces the need for the CAMC01 file to be created and/or sent.	N/A	N/A	CAMS01 TRAN ACK (Response only)
CAM confirmation file	CAMC01	CAMC01	CAMC01
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	CAMC01 TRAN ACK (Response only)
CAM return file	CAMR01	CAMR01	CAMR01
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	CAMR01 TRAN ACK (Response only)
Encryption key update request (CommonLine and CAM)	CL COMM UPDATE	CL_COMM_UPDATE	CL COMM UPDATE
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	CL COMM UPDATE TRAN ACK (Response only)
Request Document, CR:C Version 1	CRC01 REQUEST	CRC01_REQUEST	CRC01 REQUEST
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	CRC01 REQUEST TRAN ACK (Response only)
Response Document, CR:C Version 1	CRC01 RESPONSE	CRC01_RESPONSE	CRC01 RESPONSE
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	CRC01 RESPONSE TRAN ACK (Response only)

Description	Pop3 File Type Component Of Subject Line	FTP File Type Component Of Filename	DTSRequestPayloadType and/or DTSResponsePayloadType SOAP Header Value
Disbursement Roster Document, CR:C Version 1	CRC01 DISB ROST	CRC01_DISB_ROST	CRC01 DISB ROST
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	CRC01 DISB ROST TRAN ACK (Response only)
Disbursement Forecast Document, CR:C Version 1	CRC01 DISB FORC	CRC01_DISB_FORC	CRC01 DISB FORC
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	CRC01 DISB FORC TRAN ACK (Response only)
Disbursement Acknowledgement Document, CR:C Version 1	CRC01 DISB ACK	CRC01_DISB_ACK	CRC01 DISB ACK
Received by DTS Service acknowledgement. Payload can be empty	N/A		CRC01 DISB ACK TRAN ACK (Response only)
Application Document, CR:C Version 1	CRC01 APP SEND	CRC01_APP_SEND	CRC01 APP SEND
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	CRC01 APP SEND TRAN ACK (Response only)
Change Document, CR:C Version 1	CRC01 CHNG SEND	CRC01_CHNG_SEND	CRC01 CHNG SEND
Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	CRC01 CHNG SEND TRAN ACK (Response only)
Certification Request Document, CR:C Version 1	CRC01 CERT REQ	CRC01_CERT_REQ	CRC01 CERT REQ
Certification Request Document, CR:C Version 1 Received by DTS Service acknowledgement. Payload can be empty	N/A	N/A	CRC01 CERT REQ TRAN ACK (Response only)
NSLDS Lender Manifest File, Release 1*	NSL01 MFST	NSL01_MFST	NSL01 MFST
Lender Manifest Reject File, Release 1*	NSL01 MFST REJ	NSL01_MFST_REJ	NSL01 MFST REJ
Lender Manifest Unreported Loans File, Release 1*	NSL01 MFST UREP	NSL01_MFST_UREP	NSL01 MFST UREP
Generic Files **	MSC01{Desc}	MSC01{Desc}	MSC01{Desc}
Request for key fingerprint verification of PGP public key	N/A	N/A	KEY VERIFICATION 02

Description	Pop3 File Type Component Of Subject Line	FTP File Type Component Of Filename	DTSRequestPayloadType and/or DTSResponsePayloadType SOAP Header Value
Response to request for key fingerprint verification, True or False.	N/A	N/A	KEY VERIFICATION 02 RESPONSE
Request by DTS Client for any data the service may have for it	N/A	N/A	DTS RETRIEVE
Response to DTS Retrieve when there is no data available	N/A	N/A	NOTHING TO PROVIDE (Response only)

* The values described as valid payload types for this type of data are optionally supported and communication between trading partners should take place prior to sending this type of data.
** The {Desc} segment is unrestricted, however all other constraints, including the total length of the field and valid character set still apply. This is for Partner to Partner communications not formally covered by the EEAT and must be mutually agreed upon by the involved parties.

Examples

OpenPGP user ID examples

The following examples shows the OpenPGP user ID when given values for each of the components of a user ID, as detailed in the OpenPGP User ID's section.

Example 1

Organization:	School (Hunter Junior College)
ED Number:	012222
Branch campus:	01
Non-ED Branch ID:	N/A
Optional e-mail address:	N/A
OpenPGP User ID:	Hunter Junior College (CLDATA.DOE.01222201)

Example 2

Organization:	School (Larwell University)
ED Number:	013333
Branch campus:	01
Non-ED Branch ID:	EAST
Optional e-mail address:	school@learning.edu
OpenPGP User ID:	LU (CLDATA.DOE.01333301.EAST) <school@learning.edu>

Example 3

Organization:	Lender (Valley Bank)
ED Number:	809999
Branch campus:	N/A
Non-ED Branch ID:	01
Optional e-mail Address:	lender@money.com
OpenPGP User ID:	Valley (CLDATA.DOE.809999.01) <lender@money.com>

Example 4

Organization:	Lender (State Savings)
ED Number:	808888
Branch campus:	N/A
Non-ED Branch ID:	N/A
Optional e-mail address:	N/A
OpenPGP User ID:	State Savings (CLDATA.DOE.808888)

Example 5

Organization:	Guarantor (Universal Guaranty Agency)
ED Number:	825
Branch campus:	N/A
Non-ED Branch ID:	N/A
Optional e-mail address:	N/A
OpenPGP User ID:	UGA (CLDATA.DOE.825)

Email Subject line

The following examples shows the particular subject line when given values for each of the components of a subject line, as detailed in the E-mail subject line components section.

Example 1

File:	Release 5 Application Send File
Encryption:	OpenPGP
Organization:	School
ED Number:	009999
Branch campus:	00
Non-ED Branch ID:	EAST
Unique Message ID	Date (April 30, 2002) and sequence number (99)
Subject Line:	COM05 APP SEND [02] <DOE.00999900.EAST:2002043099>

Example 2

File:	Release 5 Response File
Encryption:	OpenPGP
Organization:	Guarantor
ED Number:	808
Non-ED Branch ID:	N/A
Unique Message ID	Date (October 2, 2000) and time (6:20:09:20 AM) both converted to BASE36 format
Subject Line:	COM05 RESP [02] <DOE.808:07N0DKW0E>

Example 3

File:	Version 1 CAM submission file
Encryption:	OpenPGP
Organization:	Lender
ED Number:	807777
Non-ED Branch ID:	01
Unique Message ID	Proprietary format
Subject Line:	CAMS01 [02] <DOE.807777.01:BHKL.99.KJDZ>

Example 4

File:	Version 1 CAM return file
Encryption:	OpenPGP
Organization:	Lender
ED Number:	806666
Non-ED Branch ID:	N/A
Unique Message ID	Date (April 29, 1999) and time (5:30:15 AM)
Subject Line:	CAMR01 [02] <DOE.806666:19990429053015>

Example 5

File:	CommonLine/CAM OpenPGP key update request
Encryption:	OpenPGP
Organization:	School
ED Number:	004444
Branch campus:	01
Non-ED Branch ID:	N/A
Unique Message ID	Simple sequence number in BASE36 format
Subject Line:	CL COMM UPDATE [02] <DOE.00444401:OA>

Example 6

File:	Common Record: CommonLine Request Documents Version 1
Encryption:	OpenPGP
Organization:	School
ED Number:	004444
Branch campus:	01
Non-ED Branch ID:	N/A
Unique Message ID	Date (April 30, 2002) and sequence number (99)
Subject Line:	CR:C01 REQUEST [02] <DOE.00444401:2002043099>

Example 7

File:	Release 5 Application Send File
Encryption:	OpenPGP
Organization:	Vender
VID Number:	VenderA
Routing ID:	EAST
Unique Message ID	Date (April 30, 2004) and sequence number (99)
Subject Line:	COM05 APP SEND [02] <VID.VENDERA.EAST:2004043099>

FTP filename examples

The following examples shows the particular filename when given values for each of the components of a filename, as detailed in the FTP filename components section.

Example 1

File:	Release 5 Application Send File
Encryption:	OpenPGP
Organization:	School
ED Number:	009999
Non-ED Branch ID:	EAST
Unique Message ID	Date (April 30, 2002) and sequence number (999)
File Name:	COM05_APP_SEND-02-DOE_00999900_EAST-020430999

Example 2

File:	Release 5 Response File
Encryption:	OpenPGP
Organization:	Guarantor
ED Number:	808
Non-ED Branch ID:	N/A
Unique Message ID	Date (October 2, 2000) and time (6:20:09:20 AM) both converted to BASE36 format
File Name:	COM05_RESP-02-DOE_808-07N0DKW03

Example 3

File:	Version 1 CAM submission file
Encryption:	OpenPGP
Organization:	Lender
ED Number:	807777
Non-ED Branch ID:	01
Unique Message ID	Proprietary format
File Name:	CAMS01-02-DOE_807777_01-BHKL_99_KJDZ

Example 4

File:	Version 1 CAM return file
Encryption:	OpenPGP
Organization:	Lender
ED Number:	806666
Branch campus:	N/A
Non-ED Branch ID:	N/A
Unique Message ID	Date (April 29, 1999) and time (5:30:15 AM)
File Name:	CAMR01-02-DOE_806666-19990429053015

Example 5

File:	CommonLine/CAM OpenPGP key update request
Encryption:	OpenPGP
Organization:	School
ED Number:	004444
Branch campus:	01
Non-ED Branch ID:	N/A
Unique Message ID	Simple sequence number in BASE36 format
File Name:	CL_COMM_UPDATE-02-DOE_00444401-OA

Example 6

File:	Common Record: CommonLine Request, Version 1
Encryption:	OpenPGP
Organization:	School
ED Number:	004444
Branch campus:	01
Non-ED Branch ID:	N/A
Unique Message ID	Simple sequence number in BASE36 format
File Name:	CR:C01_REQUEST-02-DOE_00444401-OA

Example 7

File:	Release 5 Application Send File
Encryption:	OpenPGP
Organization:	Vender
ED Number:	AVENDER
Unique Message ID	Date (April 30, 2004) and sequence number (999)
File Name:	COM05_APP_SEND-02-VID_AVENDER-040430999

DTS SOAP Examples

The following examples show the particular SOAP Components when given values for each of the components, as detailed in the DTS SOAP Components section.

Example 1 Request

File:	Common Record: CommonLine Request, Version 1
Source Organization:	School
Source ED Number:	004444
Source Branch campus:	01
Source Non-ED Branch ID:	N/A
Recipient Organization:	Servicer
Recipient ED Number:	700121
Recipient Branch campus:	N/A
Recipient Non-ED Branch ID:	N/A
Processing Requested:	Immediate
Unique Message ID:	UUID as defined

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
      <DTSRequestPayloadBytes xsi:type="DTSRequestPayloadBytes" xmlns="urn:org:pesc:datatransport">
          <value>1809</value>
      </DTSRequestPayloadBytes>
      <DTSRequestPayloadType xsi:type="DTSRequestPayloadType" xmlns="urn:org:pesc:datatransport">
          <value>CR:C01 Request</value>
      </DTSRequestPayloadType>
      <DTSRequestRouting xsi:type="DTSRequestRouting" xmlns="urn:org:pesc:datatransport">
          <UUID>b3ee4332-9566-4bfa-b514-b7b19a7e5c7c</UUID>
          <transmissionDateTime>10/11/2006 6:21:25 PM</transmissionDateTime>
          <sourceID>00444401</sourceID>
          <sourceIDSubCode>DOE</sourceIDSubCode>
          <recipientID>700121</recipientID>
          <recipientIDSubCode>DOE</recipientIDSubCode>
      </DTSRequestRouting>
      <DTSRequestServiceExpectation xsi:type="DTSRequestServiceExpectation"
          xmlns="urn:org:pesc:datatransport">
          <value>Immediate</value>
      </DTSRequestServiceExpectation>
      <DTSRequestSignature xsi:type="DTSRequestSignature" xmlns="urn:org:pesc:datatransport">
          <value>base64 encoded signature string</value>
      </DTSRequestSignature>
  </soap:Header>
  <soap:Body>
      <DTSRequest xmlns="urn:org:pesc:datatransport">compressed/base64 encoded data</DTSRequest>
  </soap:Body>
</soap:Envelope>
```

Example 1 Response

File:	N/A
Source Organization:	Servicer
Source ED Number:	700121
Source Branch campus:	N/A
Source Non-ED Branch ID:	N/A

Recipient Organization:	School
Recipient ED Number:	004444
Recipient Branch campus:	01
Recipient Non-ED Branch ID:	N/A
Processing Acknowledged:	Deferred
Unique Message ID:	UUID as defined


```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Header>
        <DTSResponseAcknowledge xsi:type="DTSResponseAcknowledge"
                xmlns="urn:org:pesc:datatransport">
            <value>Deferred</value>
        </DTSResponseAcknowledge>
        <DTSResponsePayloadBytes xsi:type="DTSResponsePayloadBytes"
                xmlns="urn:org:pesc:datatransport">
            <value>62</value>
        </DTSResponsePayloadBytes>
        <DTSResponsePayloadType xsi:type="DTSResponsePayloadType" xmlns="urn:org:pesc:datatransport">
            <value>CR:C01Request Tran Ack</value>
        </DTSResponsePayloadType>
        <DTSResponseRouting xsi:type="DTSResponseRouting" xmlns="urn:org:pesc:datatransport">
            <UUID>b3ee4332-9566-4bfa-b514-b7b19a7e5c7c</UUID>
            <transmissionDateTime>10/11/2006 6:21:25.830</transmissionDateTime>
            <sourceID>700121</sourceID>
            <sourceIDSubCode>DOE</sourceIDSubCode>
            <recipientID>00444401</recipientID>
            <recipientIDSubCode>DOE</recipientIDSubCode>
        </DTSResponseRouting>
        <DTSResponseSignature xsi:type="DTSResponseSignature" xmlns="urn:org:pesc:datatransport">
            <value>base64 encoded signature string</value>
        </DTSResponseSignature>
    </soap:Header>
    <soap:Body>
        <DTSResponse xmlns="urn:org:pesc:datatransport">Service created token</DTSResponse>
    </soap:Body>
 </soap:Envelope>
```


Example 2 Request

File:	Common Record: CommonLine Request, Version 1
Source Organization:	School
Source ED Number:	001111
Source Branch campus:	00
Source Non-ED Branch ID:	N/A
Recipient Organization:	Guarantor
Recipient ED Number:	123
Recipient Branch campus:	N/A
Recipient Non-ED Branch ID:	N/A
Processing Requested:	Immediate
Unique Message ID:	UUID as defined


```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <soap:Header>
    <DTSRequestPayloadBytes xsi:type="DTSRequestPayloadBytes" xmlns="urn:org:pesc:datatransport">
        <value>1809</value>
```

```
        </DTSRequestPayloadBytes>
        <DTSRequestPayloadType xsi:type="DTSRequestPayloadType" xmlns="urn:org:pesc:datatransport">
            <value>CR:C01 Request</value>
        </DTSRequestPayloadType>
        <DTSRequestRouting xsi:type="DTSRequestRouting" xmlns="urn:org:pesc:datatransport">
            <UUID> ff35ea09-b434-48e6-af3c-fe8752ac4f37</UUID>
            <transmissionDateTime>10/11/2006 6:21:25 PM</transmissionDateTime>
            <sourceID>00111100</sourceID>
            <sourceIDSubCode>DOE</sourceIDSubCode>
            <recipientID>123</recipientID>
            <recipientIDSubCode>DOE</recipientIDSubCode>
        </DTSRequestRouting>
        <DTSRequestServiceExpectation xsi:type="DTSRequestServiceExpectation"
            xmlns="urn:org:pesc:datatransport">
            <value>Immediate</value>
        </DTSRequestServiceExpectation>
        <DTSRequestSignature xsi:type="DTSRequestSignature" xmlns="urn:org:pesc:datatransport">
            <value>base64 encoded signature string</value>
        </DTSRequestSignature>
    </soap:Header>
    <soap:Body>
        <DTSRequest xmlns="urn:org:pesc:datatransport">compressed/base64 encoded data</DTSRequest>
    </soap:Body>
</soap:Envelope>
```


Example 2 Response

File:	Common Record: CommonLine Version 1 Response
Source Organization:	Guarantor
Source ED Number:	123
Source Branch campus:	N/A
Source Non-ED Branch ID:	N/A
Recipient Organization:	School
Recipient ED Number:	001111
Recipient Branch campus:	00
Recipient Non-ED Branch ID:	N/A
Processing Acknowledged:	Immediate
Unique Message ID:	UUID as defined


```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Header>
        <DTSResponseAcknowledge xsi:type="DTSResponseAcknowledge"
                xmlns="urn:org:pesc:datatransport">
            <value>Immediate</value>
        </DTSResponseAcknowledge>
        <DTSResponsePayloadBytes xsi:type="DTSResponsePayloadBytes"
                xmlns="urn:org:pesc:datatransport">
            <value>62</value>
        </DTSResponsePayloadBytes>
        <DTSResponsePayloadType xsi:type="DTSResponsePayloadType" xmlns="urn:org:pesc:datatransport">
            <value>CR:C01 Response</value>
        </DTSResponsePayloadType>
        <DTSResponseRouting xsi:type="DTSResponseRouting" xmlns="urn:org:pesc:datatransport">
            <UUID> ff35ea09-b434-48e6-af3c-fe8752ac4f37</UUID>
            <transmissionDateTime>10/01/2006 6:21:25.830</transmissionDateTime>
            <sourceID>123</sourceID>
            <sourceIDSubCode>DOE</sourceIDSubCode>
            <recipientID>00111100</recipientID>
```

```
            <recipientIDSubCode>DOE</recipientIDSubCode>
        </DTSResponseRouting>
        <DTSResponseSignature xsi:type="DTSResponseSignature" xmlns="urn:org:pesc:datatransport">
            <value>base64 encoded signature string</value>
        </DTSResponseSignature>
    </soap:Header>
    <soap:Body>
        <DTSResponse xmlns="urn:org:pesc:datatransport">Compressed/Encoded data</DTSResponse>
    </soap:Body>
</soap:Envelope>
```

APPENDIX A: OPENPGP SPECIFICS

GNU Privacy Guard specifics

At the time of this document's publication, this section contained the most recent information regarding the GNU Privacy Guard (1.4.9). Since GNU Privacy Guard is a rapidly evolving product, users should also reference the GNU Privacy Guard Website at http://www.gnupg.org.

Following are settings, options files, and command line arguments to control the encryption process. RFC 2440 contains additional information regarding OpenPGP specifics.

For use in this appendix, *localid* is the ID used to identify the user of the system, and *remoteid* is used as an example of another trading partner's ID. The key identifier can be used as part of the filename.

NOTE

Filenames are suggestions and may need to be adjusted for use with some systems.

Options file

By default, GPG supports a variety of algorithms. To see a list, use the `gpg --version command` which results in:

```
Pubkey: RSA, RSA-E, RSA-S, ELG-E, DSA
Cipher: 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH
Hash: MD5, SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2
```

If a more controlled set of options is desired, algorithms can be specified at run time or specified in the gpg configuration file. The following is a sample GNU Privacy Guard configuration file that selects which algorithms are used for ciphers, hash and compression. Any parameters in the option file may be used on the command line by prefacing them with "--".

```
# Gnu Privacy Guard OPTIONS file

# Symmetric Algorithms
# 3des, cast5, blowfish, twofish
# To force the algorithm used, uncomment the following line:
cipher-algo 3des

# Hash Algorithms
# md5, sha1, ripemd160
# To force the algorithm used, uncomment the following line:
digest-algo sha1

# Compression before encryption
# 1 = ZLIB
# 2 = ZIP from RFC1950 (default)
compress-algo 2

# Default key for use in signatures
default-key localid
```

```
# Always encrypt to this ID in addition to other recipients
# The ability to decrypt sent files is useful for troubleshooting
encrypt-to localid

# To ensure signature compatibility with other OpenPGP products
force-v3-sigs

# If using in automated systems, skip prompts
batch

# Misc / debug stuff
no-secmem-warning
no-greeting
```

Not specifying algorithms in the options file will allow the auto-negotiation features of the OpenPGP protocol to fully function.

GPG 1.4.9 provides a means of collecting the passphrase via file descriptor from the command line. Pass phrases can also be presented to the program through the control of standard in (STNDIN) and standard out (STNDOUT).) To do this in a Windows platform, reference Microsoft knowledge base article Q190351 via the Microsoft knowledge base web site.

To use the file descriptor to collect a passphrase from the command line, it must be 'piped' to the program. For example, in Windows:

```
echo myPassPhrase | gpg --passphrase-fd 0 --batch -d -v -o outPutName fileToEncrypt
```

This uses the --passphrase-fd option to collect the passphrase from file descriptor 0 which is the end of the piped 'echo' command. Keep in mind that in this case, the script now contains the passphrase. If this script were stored where unauthorized uses have access, the passphrase would be compromised. Therefore this method should only be used in highly secure automated environments.

Other schemes to store the passphrase in encrypted form in the Windows registry where the script reads and decrypts the value are possible, but beyond the scope of this document.

GNU Privacy Guard commands

The following commands execute encryption formats. The *italicized* portion of the command denotes a variable element that may be customized.

- Generating a public and private key pair

  This command generates keys and triggers an interactive mode.

  gpg --gen-key

- Exporting the public key file

  This command exports the binary *.ExportBin public* key file for transmission to trading partners.

  gpg --output *localid.ExportBin* **--export** *localid*

- Importing a public key file

  This command imports a trading partner's public key to the key ring (the trading partner's file extension may differ).

  gpg --import *remoteid.ExportBin*

- Encrypting and digitally signing a file

  This command encrypts and digitally signs a file.

  gpg --output *output.encrypted* **--sign --encrypt --recipient** *remoteid cldata.clear*

- Decrypting a file

  This command decrypts a file.

  gpg --output *cldata.clear* --**decrypt** *input.encrypted*

- Deleting a public key

  This command deletes a key from the key ring.

  gpg --delete-key *remoteid*

The following is an interactive dialog for generating keys from the GNU Privacy Guard command line arguments. The grayed areas represent lines entered manually.

```
c:\lib>gpg --gen-key
gpg (GnuPG) 1.4.7; Copyright (C) 2006 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Please select what kind of key you want:
   (1) DSA and Elgamal (default)
   (2) DSA (sign only)
   (5) RSA (sign only)
Your selection? 1
DSA keypair will have 1024 bits.
ELG-E keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user
ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"
```

```
Real name: Viktor Smith
Email address: vsmith@email.com
Comment: CLDATA.DOE.00123401
You selected this USER-ID:
    "Viktor Smith (CLDATA.DOE.00123401) <vsmith@email.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.

Enter Passphrase: myNewPassPhrase

Repeat Passphrase: myNewPassPhrase

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
+++++.+++++++++++++++.+++++.+++++.++++++++++.++++++++++.+++++++++++++++.+
+++++++++.++++++++++++++++
++++++++++++++++++++++...++++++++++++++++++++.................>.+++++...
.......................++++
+
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
++++++++++++++++++++++++++++++++++++++...++++++++++..+++++++++++++++.++
+++..++++++++++.++++++++++
++++.+++++++++++++++..++++++++++++++++++++++++..++++++++++>+++++..+++++>
+++++...........>+++++<+++
++.......................................................................
..+++++^^^
gpg: key 1E368754 marked as ultimately trusted
public and secret key created and signed.
```

Once the key pair has been created, further key customization can be done by modifying the key's preferences. This will mean that during the algorithm negotiation phase, the program will use the preferred algorithms offering some additional control when selecting which algorithms are most desirable while not blocking less desirable algorithms when left with no other choice.

Here is a sample dialog where CAST5 is removed from the list of preferred ciphers and the order of hash preference is changed to SHA512, SHA2156 and SHA1:

```
c:\lib>gpg --edit-key 1E368754
gpg (GnuPG) 1.4.7; Copyright (C) 2006 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

pub  1024D/1E368754  created: 2008-09-09  expires: never       usage: SC
                     trust: ultimate      validity: ultimate
sub  2048g/1BD265EA  created: 2008-09-09  expires: never       usage: E
[ultimate] (1). Viktor Smith (CLDATA.DOE.00123401) <vsmith@email.com>

Command> showpref
[ultimate] (1). Viktor Smith (CLDATA.DOE.00123401) <vsmith@email.com>
     Cipher: AES256, AES192, AES, CAST5, 3DES
     Digest: SHA1, SHA256, RIPEMD160
     Compression: ZLIB, BZIP2, ZIP, Uncompressed
     Features: MDC, Keyserver no-modify

Command> pref
[ultimate] (1). Viktor Smith (CLDATA.DOE.00123401) <vsmith@email.com>
```

```
     S9 S8 S7 S3 S2 H2 H8 H3 Z2 Z3 Z1 [mdc] [no-ks-modify]

Command> setpref s9 s8 s7 s2 h10 h8 h2
Set preference list to:
     Cipher: AES256, AES192, AES, 3DES
     Digest: SHA512, SHA256, SHA1
     Compression: ZIP, Uncompressed
     Features: MDC, Keyserver no-modify
Really update the preferences? (y/N) y

You need a passphrase to unlock the secret key for
user: "Viktor Smith (CLDATA.DOE.00123401) <vsmith@email.com>"

Command> quit
Save changes? (y/N) y
```

GPG can also generate RSA public and signature keys. DSA allows for signing keys of UP TO 1024 bits. While this is considered sufficient strength for a signing key, the encryption subkey should be at least 2048 bits. The nature of RSA allows for signing keys of 2048 bits or higher. This does not necessarily indicate that RSA is stronger than DSA for signing. However, some organizations my have security policies that prefer RSA over DSA. Here is a sample dialog:

```
c:\lib>gpg --gen-key
gpg (GnuPG) 1.4.7; Copyright (C) 2006 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Please select what kind of key you want:
   (1) DSA and Elgamal (default)
   (2) DSA (sign only)
   (5) RSA (sign only)
Your selection? 5
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user
ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Viktor Smith
Email address: vsmith@email.com
Comment: CLDATA.DOE.00123400
You selected this USER-ID:
    "Viktor Smith (CLDATA.DOE.00123400) <vsmith@email.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.

Enter Passphrase: myNewPassPhrase

Repeat Passphrase: myNewPassPhrase

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
```

```
generator a better chance to gain enough entropy.
+++++
..+++++
gpg: key FC2ADC86 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   4  signed:   1  trust: 0-, 0q, 0n, 0m, 0f, 4u
gpg: depth: 1  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 1f, 0u
pub   2048R/FC2ADC86 2008-09-09
      Key fingerprint = 986A F2E5 0BD8 237F A26D  7DCD 7005 0D7B FC2A
DC86
uid                  Viktor Smith (CLDATA.DOE.00123400)
<vsmith@email.com>

Note that this key cannot be used for encryption.  You may want to use
the command "--edit-key" to generate a subkey for this purpose.
```

In the case of RSA, only the signing key is initially generated. The encryption sub-key is generated separately and allows the flexibility to select EITHER an RSA encryption key OR an ElGamal encryption key.

```
c:\lib>gpg --edit-key FC2ADC86
gpg (GnuPG) 1.4.7; Copyright (C) 2006 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

pub  2048R/FC2ADC86  created: 2008-09-09  expires: never       usage: SC
                     trust: ultimate      validity: ultimate
[ultimate] (1). Viktor Smith (CLDATA.DOE.00123400) vsmith@email.com

Command> addkey
Key is protected.

You need a passphrase to unlock the secret key for
user: "Viktor Smith (CLDATA.DOE.00123400) <vsmith@email.com>"
2048-bit RSA key, ID FC2ADC86, created 2008-09-09

Enter Passphrase: myNewPassPhrase

Please select what kind of key you want:
   (2) DSA (sign only)
   (4) Elgamal (encrypt only)
   (5) RSA (sign only)
   (6) RSA (encrypt only)
Your selection? 6
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y
Really create? (y/N) y
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
..+++++
```

```
....+++++

pub  2048R/FC2ADC86  created: 2008-09-09  expires: never       usage: SC
                     trust: ultimate      validity: ultimate
sub  2048R/BA30035D  created: 2008-09-09  expires: never       usage: E
[ultimate] (1). Viktor Smith (CLDATA.DOE.00123400) <vsmith@email.com>

Command> quit
Save changes? (y/N) y
```

PGP E-Business Server specifics

The following are settings, options files, and command line calls to control the encryption process.

For use in this appendix, *localid* is the ID used to identify the user of this system, and *remoteid* is used as an example of another trading partner's ID. The key identifier can be used as part of the filename.

NOTE

Filenames are suggestions and may need to be adjusted for use with some systems.

Options file

An options file is used to control the options within the PGP software. The following settings yielded compliant behavior during testing. Algorithms and hash selections have not been forced in the options file to allow the auto-negotiation features of the OpenPGP protocol to function. Any parameters in the option file may be used on the command line by prefacing them with "+".

```
# NAI Pretty Good Privacy E-Business Server.CFG file

# Symmetric Algorithms
#   kPGPCipherAlgorithm_IDEA = 1
#   kPGPCipherAlgorithm_3DES = 2
#   kPGPCipherAlgorithm_CAST5 = 3
# To force the algorithm used, uncomment the following line:
ciphernum = 2

# Hash Algorithms
#   kPGPHashAlgorithm_MD5 = 1
#   kPGPHashAlgorithm_SHA = 2
#   kPGPHashAlgorithm_RIPEMD160 = 3
# To force the algorithm used, uncomment the following line:
hashnum = 2

# Compression before encryption
compress = on

#   Default key for use in signatures & self-encryption
#   The ability to decrypt sent files is useful for troubleshooting
myname = "localid"

# Always encrypt to the MYNAME ID in addition to other recipients
encrypttoself = on
```

```
# Misc / debug stuff
# Verbosity 0-2
verbose=2
armor=off
```

PGP commands

The following commands execute encryption formats. The *italicized* portion of the command denotes a variable element that may be customized.

- Generating a public and private key pair

  This command generates keys and triggers an interactive mode.

  pgp –kg

- Exporting the public key file

  This command exports the binary *.ExportBin* public key file for transmission to trading partners.

  pgp –kx localid *localid.ExportBin*

- Importing the public key file

  This command imports a trading partner's public key to the key ring (the trading partner's file extension may differ).

  pgp --ka *remoteid.ExportBin*

- Encrypting and digitally signing a file

  This command encrypts and digitally signs a file.

  pgp -s -e -z passphrase -o output.encrypted cldata.clear *remoteid*

- Decrypting a file

  This command decrypts a file.

  pgp -o *cldata.clear* **-d** *input.encrypted* **-z passphrase**

- Deleting a public key

  This command deletes a public key from the key ring.

  pgp --kr *remoteid*

Following is an interactive dialog for generating keys from the NAI PGP E-Business Server command line arguments. The grayed areas represent lines entered manually.

```
Choose the public-key algorithm to use with your new key
1) DSS/DH (a.k.a. DSA/ElGamal) (default)
2) RSA
Choose 1 or 2: 1
Choose the type of key you want to generate
1) Generate a new signing key (default)
2) Generate an encryption key for an existing signing key
Choose 1 or 2: 1
Pick your DSS ‘‘master key’’ size:
1)  1024 bits- Maximum size (Recommended)
Choose 1 or enter desired number of bits: 1024
Generating a 1024-bit DSS key.

You need a user ID for your public key.  The desired form for this
user ID is your name, followed by your E-mail address enclosed in
<angle brackets>, if you have an E-mail address.
For example:  John Q. Smith <jqsmith@nai.com>
Enter a user ID for your public key:  Entity Name (CLDATA.DOE.001243)

Enter the validity period of your signing key in days from 0 - 10950
0 is forever (the default is 0): 0

You need a pass phrase to protect your DSS secret key.
Your pass phrase can be any sentence or phrase and may have many
words, spaces, punctuation, or any other printable characters.

Enter pass phrase: passphrase
Enter same pass phrase again: passphrase

PGP will generate a signing key. Do you also require an
encryption key? (Y/n) y
Pick your DH key size:
1)  1024 bits- High commercial grade, secure for many years
2)  2048 bits- "Military" grade, secure for foreseeable future
3)  3072 bits- Archival grade, slow, highest security
Choose 1, 2, 3, or enter desired number of bits: 2

Enter the validity period of your encryption key in days from 0 - 10950
0 is forever (the default is 0): 0


Note that key generation is a lengthy process.

PGP needs to generate some random data. This is done by measuring
the time intervals between your keystrokes. Please enter some
random text on your keyboard until the indicator reaches 100%.
Press ^D to cancel
100% of required data
Enough, thank you.
...******** ........******* .
Make this the default signing key? (Y/n) y
................******* .********
Key generation completed.

0 memory frags found
exitPGP: exitcode = 0
```

APPENDIX B: PAYLOAD ROUTING

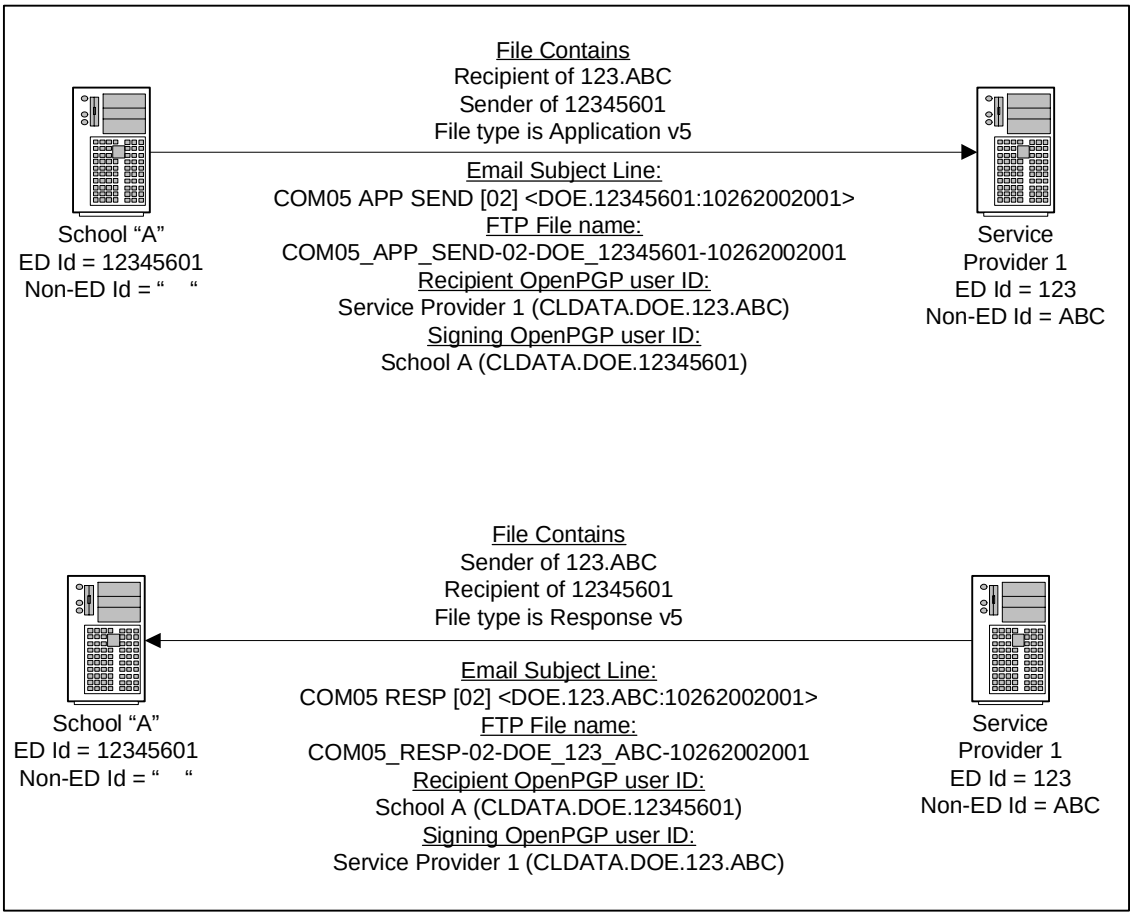
MINIMUM DATA REQUIREMENTS FOR ROUTING MESSAGES

The following data elements are required for routing entire documents from one entity directly to another entity or from one entity through one or more intermediate entities to the final destination entity.

Source Entity ID/Non-ED Branch ID
Destination Entity ID/Non-ED Branch ID
Transmitted Data File Type

- Example 1: School A encrypts and routes all of its documents directly to Service Provider 1 for processing by Service Provider 1.

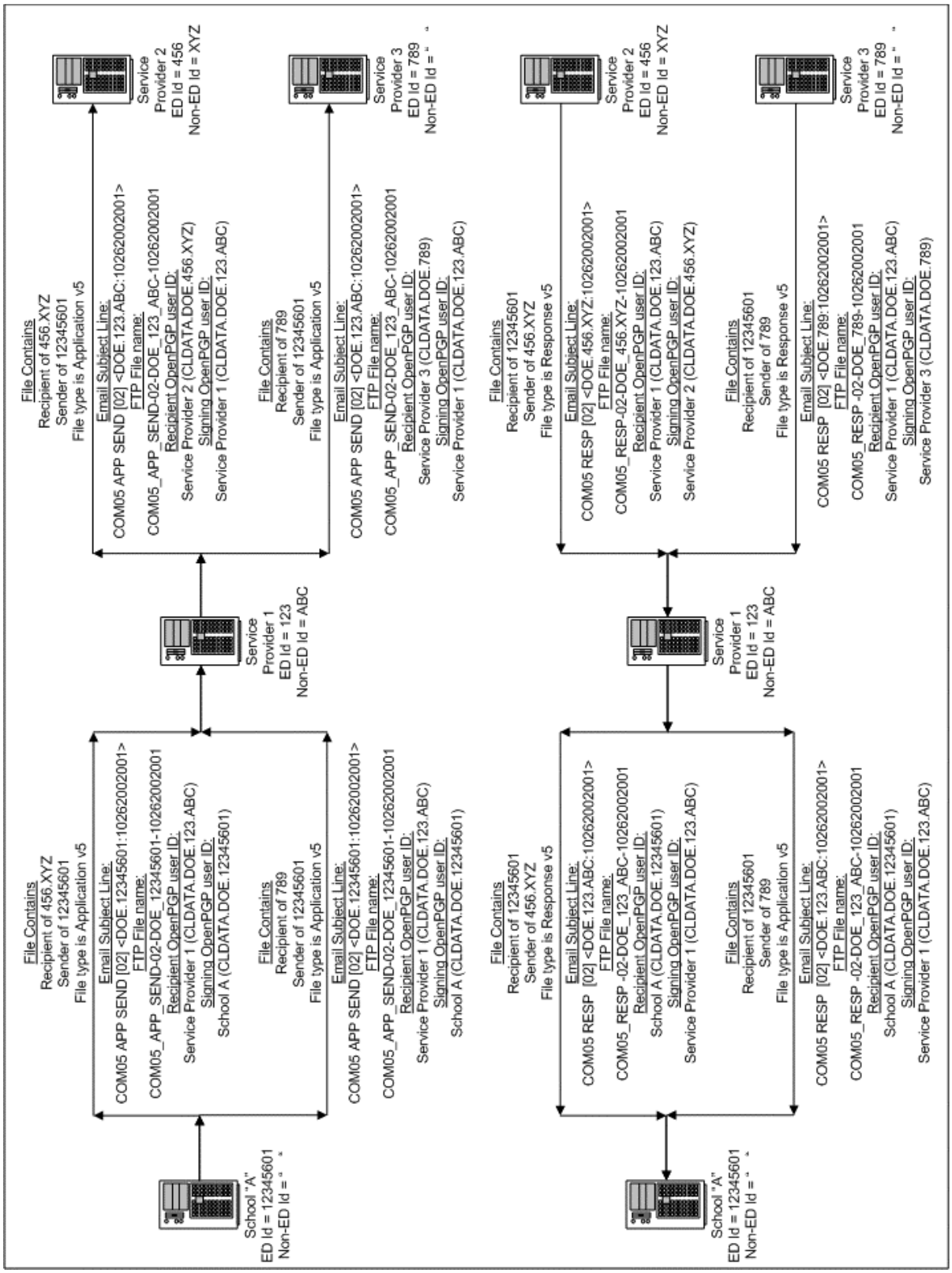
Source Entity is School A; Destination Entity is Service Provider 1. Service Provider 1 decrypts the documents upon receipt and reads the Destination Entity ID/Non-ED Branch ID from the document to identify the documents as data that should be processed in-house. These documents are moved to an in-house area to be opened for processing.



Example 1: "Simple Routing"

- Example 2: School A encrypts and routes multiple documents to Service Provider 1, who in turn retains the documents for processing in-house that have a Destination Entity ID of Service Provider 1 and routes the remaining documents on to Service Providers 2 or 3.

Source Entity for these documents is School A; Destination Entities are Service Provider 1, 2 or 3. A separate document is sent for each of the final destinations. Service Provider 1 decrypts each document upon receipt and reads the Destination Entity ID/Non-ED Branch ID. Documents with a Destination Entity ID/Non-ED Branch ID of Service Provider 1 are identified as a document that should be processed in-house. These documents are moved to an in-house area to be opened for processing. Documents containing a Destination Entity ID/Non-ED Branch ID other than Service Provider 1 will be re-encrypted, receive new routing information and will be forwarded to Service Provider 2 or 3.



Example 2: "Third Party Routing"

THIS PAGE INTENTIONALLY BLANK

APPENDIX C: SUPPORTING DOCUMENTATION

RFCs

Requests for Comments (RFCs) form a series of notes about the Internet (originally the ARPANET). The RFCs address many aspects of computer communication, focusing on networking protocols, procedures, programs, and concepts, meeting notes, and opinion.

The specification documents of the Internet protocol suite, as defined by the Internet Engineering Task Force (IETF) *http://www.ietf.org/* and its steering group the IESG, are published as RFCs. The RFC publication process plays an important role in defining and agreeing upon Internet standards.

RFC Web sites

The following sites along with many others provide a repository of RFCs:

- *http://www.rfc-editor.org*
- *http://www.cis.ohio-state.edu/services/rfc/index.html*

RFC references

The following RFCs document protocol standards discussed in this manual:

- SMTP
  821 Simple e-mail Transfer Protocol
  822 Standard for the format of ARP Internet text messages
  1869 SMTP Service Extensions
  1870 SMTP Service Extension for Message Size Post Office Protocol - Version 3 (POP3)
  1939 Post Office Protocol - Version 3

- MIME
  2045 MIME Part One: Format of Internet Message Bodies
  2046 MIME Part Two: Media Types
  2047 MIME Part Three: Message Header Extensions for Non-ASCII Text
  2049 MIME Part Five: Conformance Criteria and Examples

- FTP
  0959 File Transport Protocol
  2228 FTP using SSL or TLS security (proposed)

- OpenPGP
  2440 OpenPGP Message Format

Federal Information Protection Standards (FIPS)

Under the Information Technology Management Reform Act (Public Law 104-106), the Secretary of Commerce approves standards and guidelines that are developed by the National Institute of Standards and Technology (NIST) for Federal computer systems. These standards and guidelines are issued by NIST as Federal Information Processing Standards (FIPS) for use government-wide. NIST develops FIPS when there are compelling Federal government requirements such as for security and interoperability and there are no acceptable industry standards or solutions.

FIPS Web sites

- http://www.itl.nist.gov/fipspubs/by-num.htm

GNU General Public License (GPL)

The GNU General Public License is intended to guarantee freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it.

GPL Web Sites

- http://www.gnu.org/licenses/licenses.html
- http://www.openpgp.org/

GNU Privacy Guard

- http://www.gnupg.org/

CommonLine and CAM information

File specifications for CommonLine and CAM and documentation describing various other NCHELP initiatives can be found on the NCHELP Web site at http://www.nchelp.org/.

APPENDIX D: Considerations for "Push-Pull" implementation

In an industry survey, 80% of the respondents identified some form of an output directory. "OUT" as a specific directory name was identified by 60% of respondents. The /OUT directory structure set forth as a standard recognizes a logical structure and takes into account the majority of participants. This standard output directory ("/out") is a strong recommendation for future implementations. This does not require existing production systems to be changed; however, moving to this structure when possible is strongly recommended. Any new development and trading partner agreements allowing a "PUSH-PULL" implementation must implement the /OUT directory as defined in this manual.

THIS PAGE IS INTENTIONALLY BLANK

Appendix E: Use of Response "Tokens" in DTS

When an organization has only implemented a DTS Client there are only two models for that organization to receive processed result responses, "Immediate" and "Push-Pull". In the "Push-Pull" model, whether the DTS Client has asked for "Deferred" processing or the DTS Service can only handle "Deferred" processing the DTS response, the active acknowledgement, to the original request does not have an "official" payload. This allows the ability for the service to include a "token" as the payload that can be used by the client later at its option. The purpose of this "token" is to establish a one-to-one relationship of a processed result to a prior request.

The advantages of implementing support for tokens from both the client and service perspectives will vary greatly depending on architecture and infrastructure of internal processes. Below is a list of some perceived advantages (without going into the internal business processes/flows required to make them valid):

Client implementation	**Service implementation**
Reconcile data sent to response data received	Efficiency in answering DTS Retrieve "polling" request
Enhanced tracking for services that need to be "polled"	Tracking of data retrieved by clients

This **optional** mechanism begins if a service provider chooses to send a "token" when deferred processing will occur. If a client receives a token it could choose to use it or not, but should not generate an error if a token is encountered. This appendix will explain the overall exchange process and how it applies to utilizing a token.

The DTS Client sends the data to be processed and then needs to "retrieve" the processed result. Since the client initiates the first communication the client can know that at some point in the future the same service (end-point) should have data for it. If in the initial communication the service were to send back a "token", much like a claim check received when dropping off dry cleaning, the client could then help the service find the appropriate processed data by supplying the same "token" in the payload when "retrieving" the response.

The "token" is truly only useful to the service that created it, thus there is no need for any trading partner collaboration regarding what makes a token, its format, its content, or anything else. With exception of the following restrictions will be imposed; a maximum length of 512 bytes, and its raw representation must be text in nature. Like claim checks from different dry cleaners, you are not concerned with the color, number, or format; you just need to be able to read it and take it back to the same dry cleaner.

The client could, at its option: extract the token, store it, and send it back when requesting data from the same service. On the other hand, if the client is not concerned with helping the service or having a mechanism for creating the one-to-one relationship for itself, the token in the response could be completely ignored by the client. If the token is sent to the service by the client, it is the clients' responsibility to ensure it is the right one. The service will return a SOAP:Fault of DTS.APPLICATON (see DTS Error Handling) when a malformed token is received, just as a dry cleaner would tell you that isn't their ticket. The client could then retrieve without a token, just as you'd give your name rather than a ticket number to pick up your clothes.

Even with two trading partners taking advantage of implementing tokens, there is still the need for direct "polling" for unsolicited responses. In various business processing models there is data from Trading Partner B (the service) that Trading Partner A (the client) did not initiate the process, Trading Partner C sent data to Trading Partner B which Trading Partner A needs. The rules, values, definitions, and flows of communications between clients and services outlined in the DTS sections still apply with or without token implementation. The client has the ability to completely ignore a token as the payload of a "…Tran Ack" response but should not error if one is received.

GLOSSARY

ABSOLUTE PATH — A directory path that begins with the apparent root directory (e.g., /IS/TEST/IN).

APPLICATION SERVICE PROVIDERS — Third-party organizations that manage and distribute software-based services and solutions to customers across a wide area network (WAN) from a central data center.

APPARENT PATH — The path as reported by a File Transfer Protocol (FTP) client. The apparent path is important to the FTP client and will need to be supplied to trading partners.

APPARENT ROOT DIRECTORY — The highest directory level that an FTP client can "see." The client may not be able to access the apparent root directory and the apparent root directory is not necessarily the physical root directory (if they have been chrooted, etc.)

ASCII (American Standard Code for Information Interchange) — ASCII (pronounced ask-ee) is a code for representing English characters as numbers, with each letter assigned a number from 0—127. (E.g., the ASCII code for uppercase M is 77.) Most computers use ASCII codes to represent text, which makes it possible to transfer data from one computer to another.

ASYMMETRIC KEY – A public/private key pair that is used for encrypting, decrypting and signing data.

BASE64 ENCODING — An encoding scheme used in the Multipurpose Internet Mail Extensions (MIME) protocol.

BINARY DATA — Computer data that does not necessarily translate directly into readable text. Binary data may represent any other kind of information, such as photos or executable programs.

CBC (cipher block chaining) — The process of varying the basic encoding method to make encryption more difficult to track.

CHROOT (change root) — A method of restricting an organization's (in this case an FTP client's) view of a server's directory structure to one directory tree. Only the chosen directory tree can be seen. If the client attempts to identify the directory it is in the client will be provided a path starting from the chroot directory.

For example, in a UNIX environment, a client is chrooted to "/HOME/guarantor/g123". If the client changes the directory to "/" the client will be in "/HOME/guarantor/g123". If the client initiates a print working directory command (pwd) the client is provided a directory of "/". If the client changes the directory to a subdirectory (e.g., /archive), and initiates a pwd command, its path is actually "/HOME/guarantor/g123/archive" but the pwd command provides it "/ARCHIVE".

Chroot is advantageous in two ways:

1. Chroot allows the paths an FTP client sees to be simple because the client does not see parts of the path that do not pertain to it.

2. From a security standpoint, chroot is beneficial because the FTP client is unable to navigate to anywhere outside the "chroot" directory. This allows the workings of the system to remain off-limits. While a chroot mechanism can be defeated, it is typically more difficult to do so than it is to bypass ordinary file and directory permissions.

CIPHER TEXT — Data that has been encrypted. Cipher text is unreadable until it has been decrypted (converted into plain text).

COMMAND LINE INTERFACE (CLI) – A means of operating a computer by typing a text command at an on-screen prompt and pressing the <Enter> or <Return> key to issue the command. The computer then processes the command, displays whatever output is appropriate, and presents another prompt for the next command. This mode of interaction is common in DOS and UNIX operating systems.

COMMON ACCOUNT MAINTENANCE (CAM) — An event-driven, transaction-based reporting process for lenders, servicers, and guaranty agencies that facilitates the exchange of loan, disbursement, master promissory note, person, default aversion assistance, and claim information in a standardized electronic format.

COMMON RECORD: COMMONLINE (CR:C) — An XML-based electronic data exchange standard for FFELP and alternative loan origination and disbursement processing.

COMMONLINE — The Federal Family Education Loan Program's industry standard method of transmitting student loan data between schools and their various service providers.

CUSTOM SMTP HEADER — A string of text that is added to the e-mail header information that is not normally present and is not required by an RFC. Organizations receiving the e-mail must anticipate this type of header for it to serve a purpose. Some trading partners may choose not support custom headers.

DATA FILE - In this manual when referring to electronic data, the terms Data File, Payload, and Document are synonymous and refer to any type of CommonLine and CAM compliant file that can be exchanged between CommonLine Network trading partners.

DECRYPTION — The process of decoding data that has been encrypted into a secret format. Decryption requires a secret key (private key) or password.

DES (Data Encryption Standard) — A symmetric encryption algorithm developed in 1975 and standardized by the American National Standards Institute (ANSI) in 1981. DES uses 56-bit keys and became a standard for banking and finance applications.

DES3 (a.k.a. Data Encryption Standard 3, Triple DES, or 3DES) — See Data Encryption Standard 3.

3DES (a.k.a. Data Encryption Standard 3, Triple DES, or DES3) — See Data Encryption Standard 3.

DATA ENCRYPTION STANDARD 3 (a.k.a. Data Encryption Standard 3, Triple DES, or DES3) — A symmetric encryption algorithm that involves applying the Data Encryption Standard (DES) algorithm three times to each message, each time with a different key. The three 56-bit keys combine to effectively act like a 168-bit key, which is satisfactory security for most purposes. The primary appeal of Triple DES is that the basic DES algorithm is usually easily available, making it easy to implement.

DIGITAL SIGNATURE — A process that attaches a digital code unique to the signer to an electronic message and indicates that the owner of the digital signature wrote or authorized sending of the file. A digital signature results from the use of a private key to "sign" a message. The recipient of the electronic message can use the signer's public key to verify whether or not the digital signature is valid, and whether the message has been altered since it was signed. A digital signature is the digital equivalent of an authentic, handwritten signature. Digital signatures are not related to the CommonLine E-Signature.

DOCUMENT - See Data File

ED NUMBER (United States Department of Education Code) – A numeric code assigned by the United States Department of Education to identify an organization participating in the Federal Family Education Loan Program (FFELP). Examples include a 3-digit guarantor ID (as shown in Appendix A of the CommonLine Manual), a 6-digit lender ID or an 8-digit school ID (last 2 digits identify the branch campus). See NCHELP NUMBER for alpha numeric NCHELP assigned ID's.

ELECTRONIC SIGNATURE (E-Signature) — A method of electronically verifying a borrower's intent to sign documentation in lieu of a handwritten signature.

ELGAMAL — An encoding system that derives its security benefit from the difficulty involved in calculating discrete logarithms.

ENCODING — The process of converting data from one system of communication to another. (E.g., converting ASCII code to Base64).

ENCRYPTION — The process of encoding data in a way that prevents unauthorized viewing, especially while the data is being transmitted. Encryption renders data content unreadable to anyone except organizations that have the correct key to decrypt the data. Encrypted data is commonly referred to as cipher text.

END-OF-FILE INDICATOR — The record terminator in the trailer record of a file. The record terminator is most commonly a carriage return or line feed.

END-OF-RECORD INDICATOR — See Record Terminator.

ENTROPY — The measure of randomness in a given amount of information. In data encryption, entropy is crucial. The more random a number, the longer it will take for a computer to calculate it and consequently to defeat your encryption.

FILE TYPE FAMILY — A super-group of files (i.e. CAM, CommonLine, Meteor) that are related to an NCHELP initiative.

FTP (File Transfer Protocol) — FTP is an Internet protocol that allows file transfers (uploads and downloads) from one server to another server. For the process to work, an organization must be granted access to the desired FTP server. Once access has been given, the organization can upload files to and download files from the FTP server.

FULL-TIME INTERNET CONNECTION — A constant connection to the Internet (24 hours per day, 7 days per week).

FULL-TIME FTP SERVICE — A constant FTP connection to the Internet (24 hours per day, 7 days per week).

HASH ALGORITHM — An algorithm used in the hashing process. The hashing process (running a cryptographic hash algorithm on a file) creates a fixed length, alphanumeric string (also known as a fingerprint). Hash algorithms generate a fingerprint that is unique to the input file. SHA-1 is an example of a hash algorithm.

HOME DIRECTORY — The directory in which an FTP client is placed upon logging on to an FTP server. This directory must contain the '/IN' and '/ARCHIVE' directories and may contain the optional '/LOG' directory.

INTERMITTENT ISP CONNECTION — A connection to the Internet (using an Internet Service Provider [ISP] that is initiated and terminated by the user. An example is a dialup connection via modem. Also known as On Demand Internet Connection.

INTERNET SERVICE PROVIDER (ISP) — An organization or commercial enterprise that provides access to the Internet.

KEY IDENTIFIER — An identifier that allows the receiving organization to determine which key is to be used to decrypt a file.

KEY PAIR — A public key and its complimentary private key. In public key cryptosystems, like the PGP program, each user has at least one key pair.

KEY RING — A method of storing keys. For example, with PGP each user has two types of key rings: a private key ring and a public key ring.

KEYS — Data that is used to encrypt or decrypt a message. One type of key "locks" data and renders it unreadable. Another type of key "unlocks" the data and allows use of that data. In most digital signature programs, keys are very long prime numbers.

LZSS COMPRESSION (Lempel Ziv Storer Szymanski) — Refers to "lossless" compression for all possible data.

MIME (Multipurpose Internet Mail Extensions) — MIME is a standard system for identifying the type of data contained in a file based on its extension. MIME is an Internet protocol that allows an organization to send binary files across the Internet as attachments to e-mail messages. These files include graphics, photos, sound and video files, and formatted text documents. MIME negotiates many different operating systems and types of software.

NCHELP NUMBER (National Council Higher Education Loan Programs Code) — An alphanumeric ID assigned by NCHELP for those servicers that do not have a numeric ED-assigned ID. The length of this non-ED assigned identifier is not related to the type of servicer to which it is assigned and can range from between 3 and 8 characters in length. The appropriate value must begin with an alpha character (uppercase only - excluding special characters). See ED NUMBER for numeric ED assigned ID's.

NOTE

If you are a servicer or a service bureau and you have not been assigned an ED number, contact the NCHELP central office to determine the appropriate character value.

NON-ED BRANCH ID — An ID assigned by an organization to be utilized for internal routing unlike a Branch ID, which is assigned by the United States Department of Education.

ON-DEMAND ISP CONNECTION — See Intermittent ISP Connection.

OPEN ENCRYPTION PROTOCOLS — Encryption protocols whose algorithms are publicly available and open for review and comment.

OPEN PGP — An open standard for Pretty Good Privacy (PGP). Refer to RFC2440 (OpenPGP Message Format) for more information.

OPEN SOURCE — Open source software is a type of freely modifiable software, such as Linux. A definition and more information can be found at *http://www.opensource.org*.

OPEN STANDARD — A software standard that is publicly available and open for review and comment.

OUT-OF-BAND — As it pertains to verifying the validity of public encryption keys received, the confirmation process must take place using a different medium than the medium in

which the exchange took place. Out-of-band methods include phone, fax, and paper mail. FTP and email are not acceptable as out-of-band methods.

PASS PHRASE — A string of digits or characters providing confidential authentication of information. Pass phrases should only be known to the owner of the private key.

PAYLOAD - See Data File

PHYSICAL PATH — The full path to a directory on an FTP server. The physical path is not necessary to allow file uploading/downloading and should not be exchanged with trading partners for security reasons. Instead, a chroot method can be employed.

PHYSICAL ROOT DIRECTORY — The top-level directory on an FTP server.

PLAINTEXT — In cryptography, plain text refers to any message that is not encrypted.

POP3 (Post Office Protocol 3) — A protocol for accessing e-mail from an e-mail server. Messages are stored on a central e-mail server. Users can log on with an e-mail client and download their messages. All pending messages and attachments are downloaded at the same time. POP3 uses the SMTP messaging protocol.

PRIME NUMBER — A number that can only be evenly divided by itself and one.

PRIVATE KEY — One of two keys in a key pair. The private key is used to decrypt information and create signatures. A user's private key should be kept secret, known only to the user. See Public Key Cryptography.

PUBLIC-KEY — One of two keys in a key pair. The public key is used to encrypt information and verify signatures. A user's public key is disseminated to trading partners. See Public Key Cryptography.

PUBLIC KEY CRYPTOGRAPHY - A cryptographic system that uses two keys—a public key known to everyone and a private or secret key known only to the recipient of the message. When Organization A desires to send a secure message to Organization B, Organization A uses Organization B's public key to encrypt the message. Organization B then uses its private key to decrypt it.

PURGE — The process of deleting files from a drive or directory.

PUSH-PUSH MODEL — In a Push-Push model, Organization A and Organization B send data to each other without each other requesting the data.

PWP (print working directory) — A UNIX command used to print your current working directory.

RECIPIENT — As it pertains to the file transmissions of CommonLine and CAM files, a recipient can be a guarantor, a guarantor servicer, a school, a school servicer, a lender, or lender servicer.

RECIPIENT ID — The unique identification code assigned to the organization receiving this file. This ID can either be a NCHELP number or an ED number.

NOTE

The RECIPIENT ID is usually the same ID used to determine the encryption key to use when sending, but can be different if the recipient is receiving the file on behalf of another organization.

RECORD TERMINATOR — An indicator in each file record that identifies the end of the record. In a trailer record, the record terminator also identifies the end of the file. The record terminator is most commonly a carriage return or line feed.

RELATIVE PATH — A directory path that begins with the FTP server's current working directory (e.g., /TEST/IN). The absolute directory might be /IS/TEST/IN.

RFC (Request for Comments) — A document that defines Internet operating protocols. Despite the name, it is more a statement of agreed standards than a request.

RSA ALGORITHM — A public-key encryption technology developed by RSA Data Security, Inc. The acronym stands for Rivest, Shamir, and Adelman, the inventors of the technique. The RSA algorithm is based on the fact that there is no efficient way to factor very large numbers. Deducing an RSA key, therefore, requires an extraordinary amount of computer processing power and time.

SECRETAGENT — A program developed by AT&T that allows a user to send data in complete privacy. In addition the sending organization can transmit authentication with its messages so that the recipient can verify the message actually came from the sending organization.

SENDER ID — The unique identification code assigned to the organization sending this file. This ID can either be a NCHELP number or an ED number.

NOTE

The SENDER ID is usually the same as the SOURCE ID, but can be different if the sender is sending the file on behalf of another organization.

SERVICE LEVEL AGREEMENT (SLA) — A contract between an application service provider (ASP) and the end user which stipulates and commits the ASP to a required level of service. An SLA should specify a certain level of service, penalty provisions for services not provided, a guaranteed level of system performance as it relates to downtime or uptime, a specified level of customer support, and what software or hardware will be provided and for what fee.

SERVICE PROVIDER - An entity providing loan related services to schools or other entities throughout the loan processing cycle. Types of entities providing these kinds of services include: lenders, lender servicers, guarantors, guarantor servicers and school servicers.

SMTP/POP3 (Simple Mail Transfer Protocol/Post Office Protocol 3) — Internet protocols used to send and receive e-mail. See SMTP and POP3.

SMTP (Simple e-mail Transfer Protocol) — A protocol for sending e-mail messages. Most e-mail systems that send mail over the Internet use SMTP to send messages from one server to another; the messages can then be retrieved with an e-mail client using a protocol such as POP3.

SOURCE ID — The unique identification code assigned to the organization creating this file. This ID can either be a NCHELP number or an ED number.

SYMMETRIC KEY – A single key that is used to both encrypt and decrypt data.

SYMMETRIC KEY CRYPTOGRAPHY — An encryption system in which the sender and receiver of a message share a single, common key that is used to encrypt and decrypt the message. This differs from public key cryptography, which utilizes two keys—a public key to encrypt messages and a private key to decrypt them.

TIMEOUT — A period of time that is set for a program to wait for a response from a server before assuming that the server will not respond and considers it a failed connection.

TRIPLE DES (a.k.a. Data Encryption Standard 3, 3DES, or DES3) — See Data Encryption Standard 3.

ZIP COMPRESSION — A popular data compression format. Files that have been compressed with the ZIP format are called ZIP files and usually end with a .ZIP extension.