

PRIMEUR

End-to-End Security Solutions

for

WebSphere MQ

Primeur UK Ltd
32 Sackville Street
Royalty House
London W1S 3EA
United Kingdom
☎ +44 (0)20 3239 7456
☎ +44 (0) 20 7432 6051

Primeur Group
Corso Paganini 3
16125 Genova
Italy
☎ +39 010 27811
☎ +39 010 868 4913

www.primeur.com

January 2010

Table of Contents

Primeur’s Application Level Security Solutions	3
Data Secure Toolkit.....	4
<i>Key management and storage using PKCS#11</i>	4
<i>ICSF support</i>	4
<i>Certificate and CRL storage</i>	4
<i>Multi-buffer support for PKCS#7 APIs</i>	4
Using the toolkit.....	4
Data Secure for MQ End to End Security (DSMQE2E)	5
How the product works	6
DSMQE2E Configuration.....	6
DSMQ E2E Domain Configuration	6
Local Administration	8
<i>General Configuration Table</i>	8
<i>User Configuration Table</i>	8
<i>Queue Configuration Table</i>	9
DSMQE2E Processing.....	9
Additional capabilities of DSMQ End to End.....	10
Non-repudiation support	10
Support of Message Brokers.....	11
Audit Log	12

Table of Figures

Figure 1: Data Secure product components	3
Figure 2: DSMQ Administration Domain	7
Figure 3: Screenshot of DSMQ E2E Local Configuration Utility	9

Primeur's Application Level Security Solutions

There are two approaches to providing end-to-end cryptographic data security for WebSphere MQ messages using either digital signature (to ensure message integrity) or digital signature and data encryption (to ensure message integrity and privacy).

1. Using a cryptographic toolkit.
2. Using a product specifically designed to provide these services to WebSphere MQ™ applications.

Under their "Spazio" brand, Primeur have developed the Data Secure product which has four components:

DSTK Data Secure Toolkit: A cryptographic toolkit providing end-to-end security API calls for embedding in user applications.

FILESEC The Command line version. A cryptographic utility providing end-to-end security commands for embedding in user scripts / JCLs

DSMQ-Link Data Secure for WebSphere MQ Link: A specific point-to-point security solution for WebSphere MQ channels, which is implemented using channel exits. This is not discussed here.

DSMQE2E Data Secure for MQ End to End Security: A specific solution providing end-to-end security for WebSphere MQ applications without the need for application changes.

This is illustrated below in Figure 1

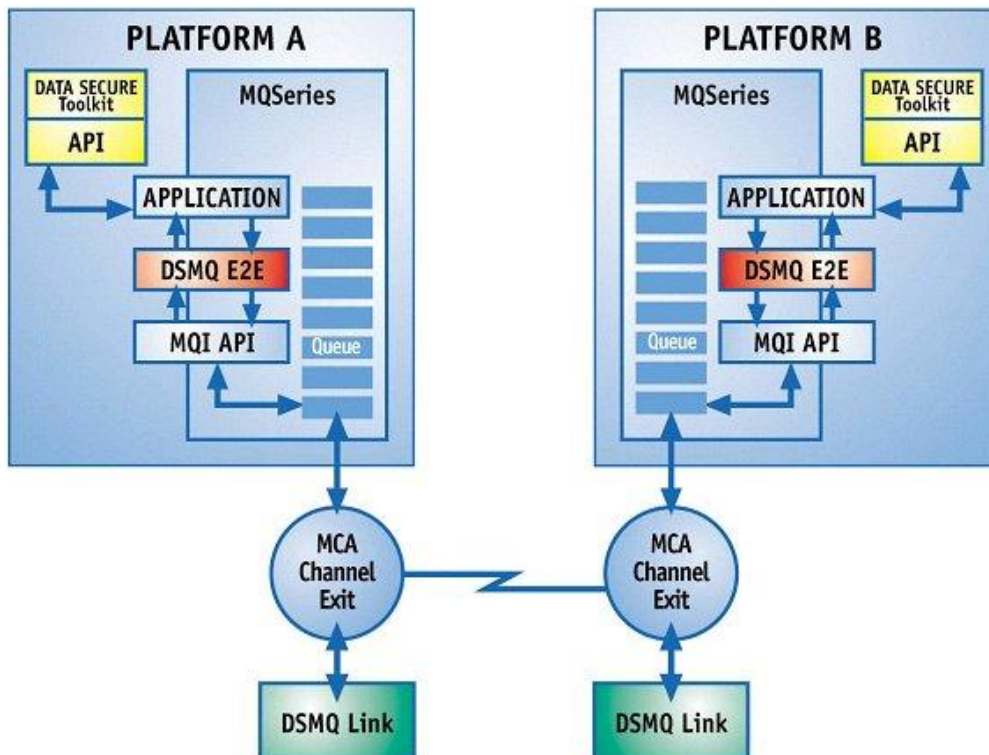


Figure 1: Data Secure product components

Data Secure Toolkit

The use of a cryptographic toolkit requires API calls to be embedded in the sending application to embed API calls to protect the data. Typically the protected data will be digitally signed to allow verification of the sender's identity, and to ensure message integrity. The data may also be encrypted so that only the intended recipient can read it. Similarly API calls must be used in the receiving application to decrypt the message (if necessary) and verify the signature.

Typically a cryptographic toolkit will be used in new applications requiring end-to-end security, or where either the sending application, the receiving application, or both lie outside the messaging and queuing environment provided by WebSphere MQ. It may also be used where archival of digitally signed messages is required for the purpose of non-repudiation checks on messages at a later date.

Using a cryptographic toolkit can be a complex task requiring the use of low-level API calls. Primeur's *Data Secure Toolkit* (DSTK) simplifies this task considerably by providing high-level function calls to simplify the creation of either S/MIME or PKCS#7 format messages thus shielding the developer from the complexities normally encountered when using cryptographic toolkits.

DSTK is available on many platforms including OS/390, Unix, Windows NT and 2000, HP NSK (Tandem), OS/2 Warp, Digital Open VMS - Alpha, and AS/400. The API set is the same on all platforms thus simplifying application development in an enterprise with multiple operating systems.

Key management and storage using PKCS#11

Asymmetric key pairs and personal certificates are stored and maintained in a PKCS#11 compliant token or "Cryptoki". By default, the toolkit uses a software implementation of the PKCS#11 token which is part of the product. Data Secure can also use PKCS#11 compliant cryptographic hardware such as that supplied by nCipher Corporation Limited (<http://www.ncipher.com>) for even higher performance and security. PKCS#11 compliant tokens from other vendors can also be used. All key management is performed in a secure and managed way within the token using the supplied token editor utility.

Primeur has tested the random number generator that is part of the supplied software token for compliance with the statistical tests for randomness specified in section 4.11.1 of the FIPS 140-1 specification. Compliance with this standard ensures that cryptographic keys are not predictable.

ICSF support

The OS/390 Integrated Cryptographic Service Facility is supported for increased performance.

Certificate and CRL storage

Public key certificates for other parties or principals are stored in a local certificate database (CDB), as are Certificate Revocation Lists (CRLs). LDAP is supported for retrieval of both certificates and CRLs that are not currently available in the local CDB.

Multi-buffer support for PKCS#7 APIs

For applications that need to create PKCS#7 format messages where the data to be processed may not fit into storage (for example large files, or data streams of unknown size), a set of APIs is available that allow the creation of these messages on a buffer-by-buffer basis. The use of these APIs is not covered in this document.

Using the toolkit

Data Secure provides APIs for the creation and reading of either PKCS#7 format or S/MIME format objects (messages). The use of these APIs is much the same regardless which format is required (though different APIs are called). Prior to using the APIs it is necessary to initialise three simple structures to define the CDB, the PKCS#11 token to be used and the associated PIN or pass phrase, and the cryptographic algorithms to be used. The APIs are callable from the C (or C++) programming language, and on OS/390 from COBOL.

Data Secure for MQ End to End Security (DSMQE2E)

DSMQE2E (also referred to as E2E) is specifically designed to provide cryptographic protection for messages sent using WebSphere MQ without requiring changes to the application. Data is protected in the form of PKCS#7 Data Objects (or messages). MQ messages may be either signed or signed and encrypted. In addition to the cryptographic protection of messages, DSMQE2E can also be used to compress message data using an LZW based compression technique. The compression of messages is particularly relevant when cryptographic processing is to be applied, as a PKCS#7 format message (or Data Object) will be larger than the clear text message from which it is created. A further benefit from compression is that the cost of compressing and then encrypting a message is typically lower than the cost of just encrypting the same message.

We support all message types including segmented messages, and Distribution Lists are fully supported.

If messages are encrypted, they are encrypted on the queues, and also in the MQ log files.

How the product works

DSMQE2E works by pre and post processing the MQ API calls. To achieve this we use a call interception technique for OS/390 batch applications and also for the MQSeries 5.2 distributed systems that we support. OS/390 CICS is supported using the MQ CICS API crossing exit (CSQCAPX). For WebSphere MQ 5.3 - and later versions - on distributed platforms we use the new MQ API crossing exit which provides us with a formally supported technique to pre and post process MQ API calls.

Processing is controlled entirely by the DSMQE2E configuration. E2E will only perform its processing on messages when both the logged on user and the queue being opened have appropriate entries in the configuration.

DSMQE2E Configuration

The active local configuration is stored in the form of three tables. These tables are stored as persistent messages on three MQ Series queues, one message for each row in the table. We use the Correlation Id to rapidly locate the correct message thus ensuring efficient access to the configuration data. All configuration data is held as string data in CCSID 819. The details of the contents of these queues are described in section Local Administration below.

The product can be administered in one of two ways. The first of these which is delivered with DSMQ release 1.10 is more suitable for production use or large configurations where change control is of paramount importance, while the second (original) method is more suitable for small installations or sites where administrative responsibility for each queue manager falls to different teams.

DSMQ E2E Domain Configuration

The *domain administration model* which is delivered with DSMQ 1.10 is based on an easy-to-use scripting language that can be used to create/deploy DSMQE2E configuration document describing the configuration of DSMQE2E on all queue managers belonging to the same administration realm.

We anticipate this model will be used for:

- Large production environments with large number of queue managers (more than 5) and/or a relatively small number of DSMQE2E guarded queues per queue manager (more than 20)
- Environments where the usage of WMQ client connections is not allowed/suitable
- Environment where a command line interaction with the system is the only available options (e.g. Unix boxes accessed with telnet protocol)
- Environment where change management policies mandate the usage of an external versioning system (such as CVS, IBM Rational ClearCase or Microsoft Visual SourceSafe) to keep track of deployed configurations

The configurations for a domain are created in the form of "scripts" held as "flat files" which may be stored in the change control system of the customer's choice.

The language chosen for the implementation of these scripts needs to be one for which an interpreter is widely available on all the relevant platforms. If the platforms involved were just UNIX systems, then the obvious candidates for the implementation would be scripting languages such as Perl, Python, or Tcl as these are widely known and understood, and available for all UNIX systems. Unfortunately, given the wide range of platforms on which WebSphere MQ runs, these do not appear at first glance to be suitable, as these languages are not typically available on (for example) z/OS.

There is a suitable candidate in the form of Java: The Java interpreter can be found on almost any platform, but Java of itself is not a scripting language but a programming language. Python, on the other

hand, is a suitable scripting language. What is needed is a scripting language implemented in Java: Jython is an implementation of the Python scripting language that is written in 100% Pure Java. This leverages the broad platform availability of Java and adds the speed of development of the Python scripting language without requiring a native Python interpreter.

With reference to Figure 2 below we will walk through the main data flows involved in DSMQ Domain Administration.

1. A DSMQ domain configuration script is created by an administrator, using a normal text editor, and is then deployed to target queue manager using DSMQCfgMgr tool (1). As part of the deploy operation a copy of the configuration is centrally kept on a configuration repository queue hosted at the DSMQ Configuration Manager queue manager. When deploying a new configuration the administrator can optionally digitally sign the configuration document.
2. Using normal WMQ addressing techniques a copy of the configuration is routed via transmission queues (optionally exposed through remote queue definitions) to the target queue managers, see (2) and (3).
3. At each of the DSMQ Domain Member queue managers managed the domain configuration document will be received by the DSMQ Command Server component – either a WMQ triggered application or a long running process, depending on platforms (4) – and stored in an optimised indexed way on a set of local configuration queues (5).
4. If configured to do so, DSMQ Command Server will also check the existence of a valid digital signature on the configuration document, against a list of authorized senders (i.e. authorized administrators).

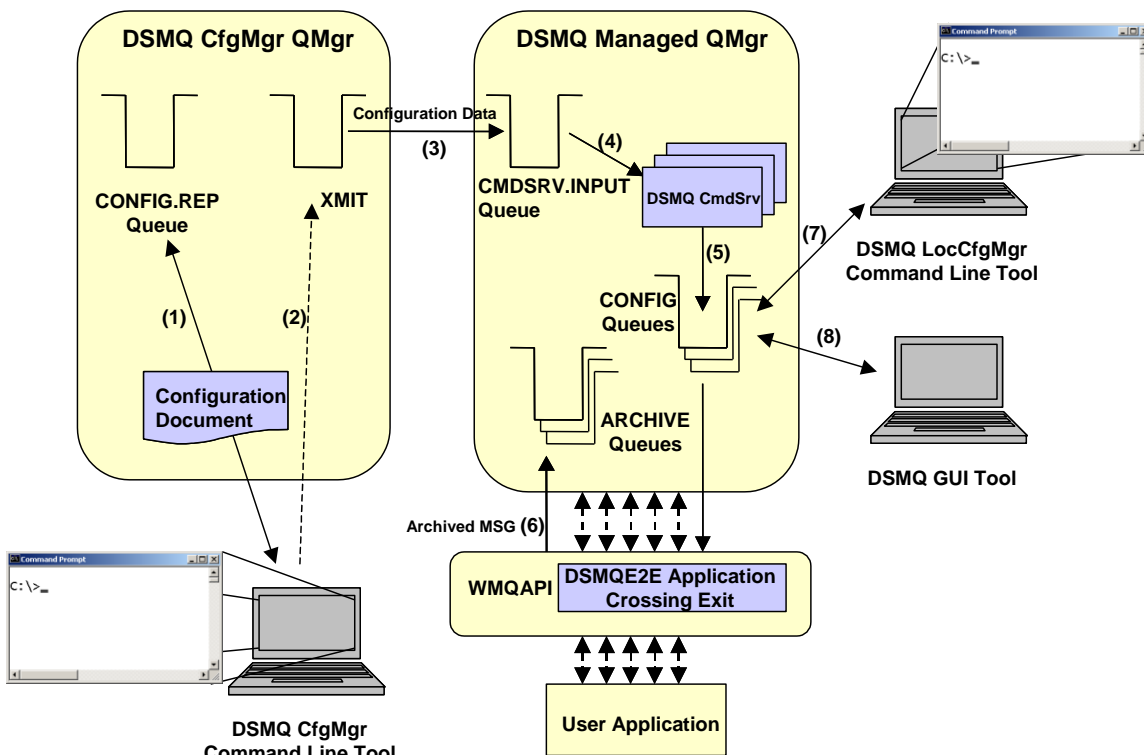


Figure 2: DSMQ Administration Domain

Strictly speaking, the DSMQ scripting is not a language in itself, but rather a set of Java classes with an interface simple and coarsely grained enough to be suited for invocation from a java scripting language, by any administrator with basic scripting skill and possibly no java specific background. While almost any Java based scripting language (e.g. Rhino, or JACL) can be used with these classes, we have chosen to

implement some of the administrative tools in Jython, and configuration export files for storage in external version control systems such as CVS are generated in Jython syntax.

The classes available for building a DSMQ E2E Domain Configuration are shown in the table below:

ClassName	Description
DSMQDomCfg	This class is a DSMQ configuration
DSMQQMgr	This class stores all definitions of a single QMgr. Can be a direct mapping to the general queue configuration. But in advance case it can keep track of Links connected, users and their roles
DSMQUser	This class keeps track of the identity of single users with information on how they are known at each Queue Manager
DSMQQueue	This class keeps track of the way a queue is accessed by the owning queue manager and by other queue managers in the configuration domain
FireForgetPattern	This is a helper class used to model fire and forget messaging patterns, It can handle a couple of connected QManager-Queue enforcing consistency among parameter settings at the two ends of the link, shielding and simplifying the specification of attributes and certificates. A WMQ cluster-aware variant of the class is available as well
RequestReplyPattern	This is a helper class that builds on top of the previous one when the underlying messaging paradigm to model is request reply. A WMQ cluster-aware variant of the class is available as well

Local Administration

A local GUI configuration tool written in Java is supplied as part of the product that is used to manage the configuration tables. Utilities are also supplied to import or export configuration data for use on another system, thus allowing a limited form of remote administration. The tool may be used for viewing a configuration created using the domain administration model. It may optionally be disabled.

A screen shot of the configuration tool is shown in Figure 3 below.

The configuration is held on three tables whose basic content is described below:

General Configuration Table

This queue holds a single row or message that specifies the CDB location.

User Configuration Table

This queue contains information on the PKCS#11 token to be used, and the name of the private key to be used to sign messages. When the application connects to the queue manager, the logged on user identifier is used as the key to locate the appropriate entry. If no user configuration entry is found when the application connects to the queue manager, no E2E processing is done for any queues opened by the application.

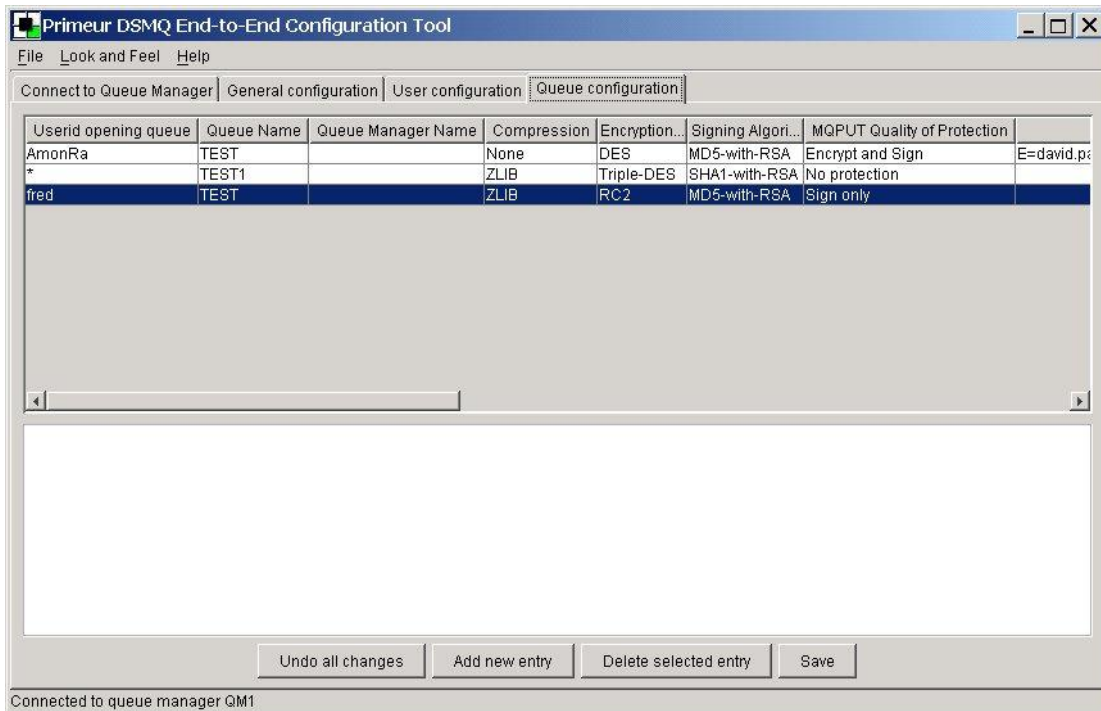


Figure 3: Screenshot of DSMQ E2E Local Configuration Utility

Queue Configuration Table

This queue defines the processing to be performed by E2E. It is keyed on the user name (logged on user identifier) opening the queue, and the queue name and queue manager name specified in the MQ Object Descriptor (MQOD). A wild card in the form of an "*" character is allowed for the user name.

For MQPUT an entry specifies:

- Whether or not the message is to be compressed
- The level of protection to be applied to the message (None, Sign, or Sign and Encrypt)
- What encryption algorithms to use if cryptography is to be used.
- The distinguished name of the recipient for whom the message is to be encrypted

For MQGET an entry specifies:

- The policy to applied when getting messages. This specifies what messages will be accepted based upon whether and how they were processed by End-to-End Security when the message was originally MQPUT by the sender.
- Optionally a list of distinguished names which specifies who is allowed to have sent the message.

The queue configuration data is read when the queue is opened by the application.

DSMQE2E Processing

When E2E intercepts an MQPUT issued by an application, it checks whether E2E processing is to be performed for messages put to this queue based on the configuration data read when the queue was opened. If E2E processing is not to be performed, the MQPUT call proceeds normally.

Data that is being sent by the application will have a specific format (for example it might be character data (MQFMT_STRING) that is to be converted from the sending queue manager's CCSID to the receiving queue manager's CCSID when the MQGET API is issued with the MQGMO_CONVERT option). When data is compressed or has cryptographic processing applied to it the resulting data will be binary data and will be a different length from the original. It is possible to configure channels to do data conversion as the data passes through the MQSeries network, but the data that we send after processing will not survive such conversion. For this reason, we preserve the original message format from the MQMD at the point where the original application puts the message, and set the message format to MQFMT_NONE.

The original message format, CCSID, encoding and other information such as the original message length and the actual E2E processing that has been applied are saved in our own message header (the Data Secure Message Descriptor or DSMD) which is the first part of the data we actually put to the queue. This message header is followed by the data resulting from the compression and, or cryptographic processing by E2E. Data that is less than 100 bytes is not compressed by E2E regardless of the configuration.

Any errors in processing are notified to the application using the completion code and reason code in the normal way. The reason codes that E2E uses are outside the range used by WebSphere MQ.

For MQGET processing, as with MQPUT processing, if no E2E processing is to be performed, then MQGET processing is allowed to proceed normally.

If the message on the queue was not processed by DSMQE2E, and the MQET policy allows this, then the message is returned normally and any data conversion requested by the application is performed. This allows the receiving application to process messages that were sent from applications for which DSMQE2E is not configured, and from those for which it is. A staged roll out of E2E within the enterprise is therefore simply enabled.

If the MQGET policy only allows messages that are signed and encrypted (for example), and the message on the queue is not signed and encrypted, the message from the queue is returned to the application without the removal of our header (if present) and without decryption, decompression or data conversion. The completion code and reason code are set to indicate this.

If the message on the queue is signed or signed and encrypted, and the distinguished name of the sender does not match one of the authorised senders (if specified), the message is returned to the application as it is on the queue (as above). The completion code and reason code are set accordingly.

Otherwise, if the data was signed or signed and encrypted, the P7GETD() API call is used to decrypt the data if necessary and to validate the signature. If this fails (for example the signature is not valid) the data from the queue is returned as above, and the completion code and reason code are set. The data is then decompressed if necessary.

The DSMD header is removed and the original message format, encoding and so on are restored in the MQMD. Finally any data conversion requested by the application is performed.

Additional capabilities of DSMQ End to End

The techniques described above allow an end to end data security solution to be delivered for WebSphere MQ that provides the services of message integrity and proof of origin (by using a digital signature) and if so wished message privacy by means of data encryption.

On their own however, these techniques may not provide an adequate solution for meeting regulatory or other requirements for after the event non-repudiation, the provision of audit trails, or to support the use of a message broker.

Non-repudiation support

In situations where messages are being exchanged between parties who do not necessarily trust one another, it can be useful for the recipient of a message to be able to prove at a later date that the other party sent a particular message. By using a digital signature on the message the identity of the sender and the authenticity of the message can be confirmed, and therefore the sender of the message is unable to deny (or repudiate) the sending of that message.

If the recipient of a digitally signed or a digitally signed and enveloped (encrypted) message wishes to use the digital signature for this purpose, then they need to archive the message in its signed or signed and

enveloped (encrypted) form. If this is done, should a dispute arise at a later date, the recipient can retrieve the message from the archive and check the signature again. This is not the only requirement for ensuring that messages cannot be repudiated, but it is a necessary one.

There are a number of other requirements for the process of non-repudiation. The most important are:

- A suitable legal framework exists to allow the use of digital signatures for this purpose. This may take the form a contract, or be part of national or supra-national legislation.
- If necessary, the communicating parties have agreed to the use of digital signatures for this purpose.
- If required by the legal framework (this is normally the case), the certificates used for checking the signature should include key usage of "non-repudiation".

Non-repudiation is normally only necessary when receiving signed or signed and encrypted messages from another organisation as might occur in a Business to Business or Business to Consumer message exchange. It is not typically required for messages moving within the organisation where the digital signature is used purely for the purposes of authenticating the message sender and that the message has not been tampered with while in transit.

Because only some messages need to be archived for non-repudiation purposes, it is not appropriate for DSMQ End-to-End Security to archive all signed or signed and enveloped message that are received. Typically the requirement will be to archive messages retrieved from a specific queue or queues.

Each customer will typically have unique requirements for archival: Some will want to archive messages to an external database, while others will wish to write these messages to some other form of external storage. For this reason, we write the archive messages to an MQSeries queue (QLOCAL, QALIAS, or QREMOTE) whose name is part of the queue configuration of DSMQ End-to-End Security. For example: We might wish to archive all signed or signed and enveloped messages read from the queue MYAPP.QUEUE. In this case we would modify the queue configuration in DSMQ E2E to specify an archive queue name of MYAPP.ARCHIVE.QUEUE.

If this is done, then once DSMQ End-to-End security has successfully validated the signature on the message, and if so configured has confirmed that the message sender was authorised, the message as it was read from the queue is written to the archive queue. Put in another way, the message written to the archive queue includes the DSMD header and the payload is still in the form of a PKCS#7 Signed or Signed and Enveloped data object or objects.

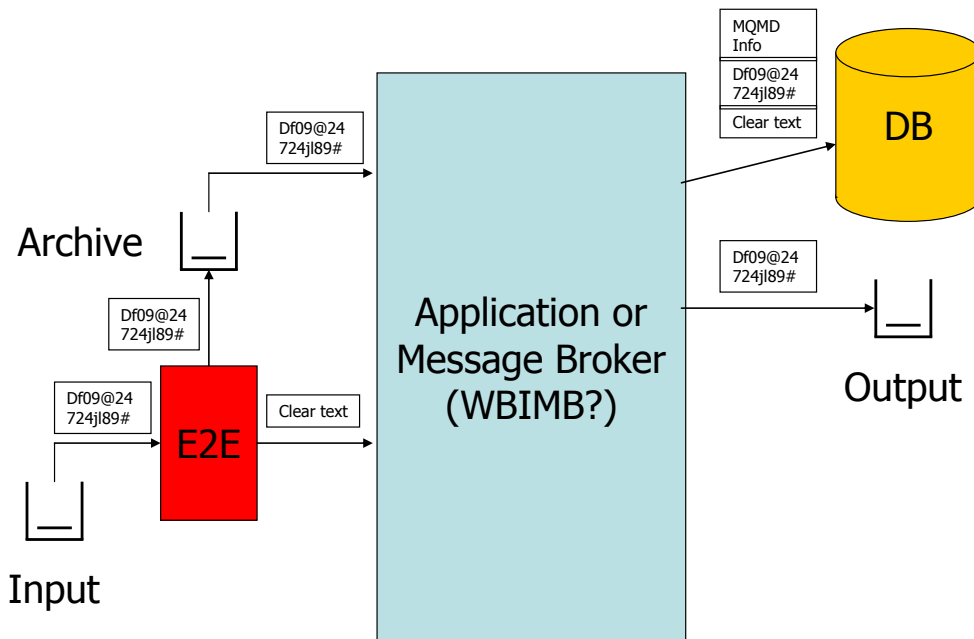
The message is written to the archive queue using MQPMO_SET_ALL_CONTEXT, so that all the information from the original message descriptor is preserved. For this reason, the user running the application reading the messages requires appropriate authority to use "set all context" on the archive queue.

The archive messages are MQPUT to the archive queue with the same syncpoint options that the user application was using to get the messages from the application queue. The archive function does not issue any commits or rollbacks.

The messages may then be read from the archive queue by the same (or another) application and saved to a database for non-repudiation purposes (where the signature is re-validated "after the event" in the event of a dispute).

Support of Message Brokers

This archive facility described above allows solutions to be developed for message brokers where it is wished for the broker to inspect the "clear text" message and yet route the original signed (or signed and encrypted) message through to the final destination. It also allows more complex solutions to be developed where the original (protected) message is archived along with the plain text, and also to transform the message and record the clear text and protected content that has been sent on to the final processing node.



Audit Log

Regulatory requirements, internal practices, or similar often require that an audit log be kept of processing applied to data. This might only require that exception conditions be logged, or that all processing be logged. It might be required for all messages processed on all queues, or just for some queues.

For this reason, DSMQ End-to-End Security may be configured on either a global basis or a queue by queue basis to write an audit log of its activities. This audit log may record either all E2E related processing, or just those E2E actions that resulted in an error condition.

The audit records could be written in a platform specific manner (such as for example using SMF records on z/OS), or a platform neutral manner using a format such as XML written to a flat file. We believe that a platform neutral solution such as XML provides a portable solution which has the advantage that it can be read by people without requiring any special processing, and also by programmes.