Specification for FoxTalk™ TCP/IP Protocol Version 1.1

Computer Projects of Illinois, Inc.
475 Quadrangle Dr. Suite A
Bolingbrook, IL 60440
(630) 754-8820

* * * THIS PAGE LEFT INTENTIONALLY BLANK * * *

Table of Contents

1 Overview

The FoxTalk™ Protocol was developed by CPI to interface our various client software products with our OpenFox™ Message Switch in a consistent manner. CPI considers the protocol open to implementation by anyone, and will freely release the specification. There are no royalty or license charges for use of the FoxTalk™ Protocol.

The FoxTalk™ Protocol represents an application-to-application protocol for use over a TCP/IP communications session. The protocol introduces a method for the client and server to negotiate session parameters at startup and specifies a formatting standard for delineating data within the TCP/IP data stream.

TCP/IP provides a connection-oriented data stream for applications to communicate. The low level drivers will guarantee that data will arrive at the destination end of an established session in the order it was sent by the originating side. It will also guarantee that data is successfully delivered before it is removed from the originators outbound buffer. The application level must solve all remaining communications issues. Below is an overview of each communications challenge and how it is addressed within the FoxTalk™ Protocol.

1.1 Connection Oriented

The FoxTalk™ Protocol specifies that an open TCP session is maintained at all times to allow the smallest possible delay in communications. The law enforcement environment involves unsolicited messages flowing at any time in either direction, from the client to the switch or vice versa. Frequently these messages contain time sensitive information such as hit requests or dangerous weather notifications. Maintaining an open communications link insures that these message may flow immediately, inbound or outbound.

1.2 Message Framing

Since TCP/IP provides a stream format for data exchange, the applications must use a consistent method to delineate the beginning and ending of a message within the data stream. The FoxTalk™ Protocol uses a framing technique that is very similar to the NCIC-2000 framing method. The protocol specifies a start and stop pattern identical to NCIC-2000 framing, and a frame length which is extended to 32-bits from the NCIC-2000 standard of 16-bits. The hybrid frame allows more flexibility for large frames while maintaining a familiarity to anyone with experience in the law enforcement market.

1.3 Content Negotiation

In modern law enforcement messaging environments the message switch is frequently at the center of a network of dissimilar communicating applications. Often times the end applications are developed by many different vendors or are going through upgrades where a common vendor may have multiple versions of an application in the field at once. These different clients will typically have different capabilities. FoxTalk™ will allow clients of different capabilities to communicate successfully without relying on OpenFox™ system administrators to make configuration changes to each device. The session parameters will be negotiable by the client after connecting. The features that may be negotiated include:

- Maximum frame length allowed
- Use encrypted data or plain text
- Allow binary object transmissions (such as images) or not, and if so specify encoding technique
- Specify new-line sequence for text blocks
- Specify maximum allowed idle time and default timeout

This negotiation technique allows image capable and non-image capable devices to specify their preference on session establishment. If a non image capable device is later upgraded to be image capable the new software version simply negotiates image capability with OpenFox™, thus obviating the need for an administrator to change the device configuration tables. The maximum frame length will allow multiple clients to negotiate different block sizes without relying on individually configuring each device differently. The ability to request unencrypted data will be useful in a site where the network routers perform encryption. In such a case the software encryption only adds overhead and may be safely disabled. Making this parameter negotiable allows client developers to add flexibility to their product's configuration. Finally, the ability of the client to specify his preferred new-line sequence removes ambiguity from parsing messages specifically for new-lines and guarantees an acceptable presentation format on the client's system.

1.4 Application Acknowledgement

Since an application program could fail after the confirmed arrival of data at the TCP layer but before the application has read the data out of the TCP receive buffer the possibility of data loss exists unless applications send acknowledgements to each after successfully receiving data. FoxTalk™ specifies a simple application acknowledgement to secure the communications link from data loss.

1.5 Connection Maintenance

Because FoxTalk™ specifies that an open TCP session exist at all times when an application is ready and able to exchange data, a simple heartbeat mechanism is used to catch link or application failures in a timely fashion. The danger in not using heartbeats lies in the client's ability to detect that it is no longer connected to the switch. For example, many workstations function primarily as receivers (such as unattended printers) and don't have a large volume of transactions initiated to the switch. Since these devices rarely send any data they can't rely on send failures to detect a failed link. In these cases, if a half session failure occurs it will be unlikely that anyone will notice; meanwhile the workstation may not be receiving messages that are queued on the message switch. Heartbeats will allow an unattended application to recognize that the link has failed and automatically re-establish connectivity.

1.6 Frame Exchange Methodology

The FoxTalk™ protocol functions by building frames of information and exchanging them. This is what guarantees that every message sent receives some sort of acknowledgement from the other side – every meaningful task is an exchange of frames. For example, session parameters are negotiated after startup by exchanging Connect Message frames. An idle session is maintained through a Heartbeat exchange. A data message is delivered through the exchange of one or more data message frames and a corresponding ACK or NAK frame.

* * * THIS PAGE LEFT INTENTIONALLY BLANK * * *

2 Protocol Flow

The FoxTalk™ protocol requires that a device establish a TCP/IP session with the OpenFox™ message switch and negotiate session parameters to create an open session. Once the session is open, messages can be initiated at any time in either direction. Every message must get a response from the recipient before the next message is sent. If the maximum idle time is reached the client must send a heartbeat. The OpenFox™ will respond by echoing the heartbeat back to the client. Below is a description of each of the required steps in detail.

2.1 TCP Connect

The client should establish a TCP connection with the OpenFox™ IP address and published port number. These values are unique to each account and must be determined by each account's system administrators. If a TCP connect fails the client should wait a **minimum** of 30 seconds before attempting another connection request. Failure to delay between connection requests can cause a condition resembling a denial of service attack to exist on the message switch network and must be avoided. After a successful connection has been established the client should send a FoxTalk™ connect message as the first step in negotiating session parameters.

2.2 Connect Message

After accepting a TCP connection from a client the OpenFox™ message switch will require that the next message received be a FoxTalk™ connect message. The client should choose from the available session parameter choices, construct a connect message, and send the message to OpenFox™. OpenFox™ will adjust the values in the connect message to values that OpenFox™ can support and that are required by the context in which OpenFox™ and the client are communicating. OpenFox™ will then return the connect message with the altered parameters. The client must be able to process the parameters as they are returned from OpenFox™. If the client is unable or unwilling to do so it should disconnect from OpenFox™ and notify the user that it is unable to accept the parameters required by OpenFox™. If the client is able to accept the parameters then a successful communications session has been established. If encryption has been negotiated, then the client and server must immediately exchange a series of Key Negotiation messages to set the encryption key for the session. Otherwise, message flow in either direction may commence immediately. Please see the detailed documentation of the connect message options in section **5 – Connect Message Details**.

2.3 Encryption Negotiation

Upon negotiating an encrypted session, the server and client must exchange a series of frames to securely choose a random encryption key for this session. FoxTalk™ sessions use a random 128-bit AES key for each session, and then key lives only as long as the session. To set this key a series of Key Negotiation frames (Type 'K') are send back and forth as follows:

- Servers sends the client a Server Nonce frame
- Client uses 2048-bit public RSA key of server to encrypt a message contained the server nonce, a client nonce, and an AES key chosen at random by a cryptographically secure pseudo-random number generator.
- The server sends the client nonce back to the client having encrypted it with the randomly chosen AES key

This sequence of messages prevents replay attacks (through the use of the server nonce), allows the client to authenticate that he is actually talking to the OpenFox™ message switch server (one-way authentication provided by the RSA key), and allows the client to detect a man-in-the-middle by receiving his client nonce back from the server. Also, since the client has to use the randomly chosen AES key to decrypt the client nonce, the client can be sure the server has set the AES key for the session properly. Please see detailed documentation of the Key Negotiation sequence in section **6 – Encryption**.

2.4 Device Identification

In the case of the client being an OpenFox™ Desktop software client, the OpenFox™ requires that the desktop software send its registered and encrypted license file to identify the device. This happens in the device identification step, where the Desktop software uses a Type 'I' message (Identify) to transmit the license file received during registration to the OpenFox™ . It should be noted that this registration file is encrypted with an AES key that only the server knows, so that this data is just an opaque (and unreadable) data blob to the client. Upon receiving this file OpenFox™ will allow the Desktop session to begin normal message exchange. Please see detailed documentation of the Device Identification phase in section **7 – Device Identification**.

2.5 Regular Messaging

After the connection negotiation has finished messages may flow in either direction. A message originator must receive an acknowledgement from the other side before sending another outbound message. For example, if the client has 4 messages to send to OpenFox™, it must send one and then wait for OpenFox™ to respond with a FoxTalk™ acknowledgement. Then it may send the second message (and so on). In the event that a message's length would cause a single FoxTalk™ frame to exceed the negotiated maximum frame length it must be broken into multiple frames. There is no acknowledgement between frames. A series of frames are simply built with all but the last having an End-Of-Exchange code of 'N' in the header. The last frame should have an End-Of-Exchange code of 'Y'. The Exchange ID field should be identical for each frame of the message. There is no need to worry about the frames arriving out of order at the other end since the low level TCP drivers will insure that that doesn't happen. After receiving the final frame of a multi-frame message the recipient will send a FoxTalk™ acknowledgement. If no acknowledgement is received by the sender, then the entire message must be retransmitted. Please note that a message originator may choose to send a message as multiple frames even if the individual frame lengths don't reach the maximum negotiate frame length. In other words, the message originator may choose to break a message into frames of any length up to the maximum negotiated frame length. The breaks may occur at any place in a message.

2.6 Idle Line Maintenance

In the event that a session exceeds the negotiated maximum idle time it must send a FoxTalk™ Heartbeat to OpenFox™. OpenFox™ will immediately echo the heartbeat back (so that the peer can verify that the connection is still alive). If an application sends a heartbeat to OpenFox™ but does not receive a heartbeat back within the negotiated default timeout it should consider the link dead and close the connection.

2.7 Connection Closure

An active FoxTalk™ session may be closed by either side at any time. An application using FoxTalk™ must be able to handle a connection being closed. The proper action upon receiving a close is wait at least 30 seconds and then attempt to re-connect. If the re-connect attempt fails, wait another 30 seconds and retry (and so on) until the connection is successful.

* * * THIS PAGE LEFT INTENTIONALLY BLANK * * *

3 Message Framing

As covered in the overview, the FoxTalk™ Protocol makes use of a basic message frame that is very similar to the NCIC-2000 framing technique. The exception is that FoxTalk™ specifies a 32-bit frame length field (as opposed to the 16-bit NCIC-2000 standard). Within the FoxTalk™ frame is a header and payload section (message content). Below is a table depicting the FoxTalk™ frame.

Element	**Description**
Frame Start Pattern	32-bit unsigned integer in network byte order having the hexadecimal value `FF00AA55`
Frame Length	32-bit unsigned integer in network byte order that contains the overall length of the frame, including the start pattern, frame length field, protocol header, frame payload and stop pattern. This value must be at least as large as the framing overhead and must be no larger than the maximum negotiated frame length.
Frame Header	FoxTalk™ Header – documented below
Frame Payload	Variable length frame content. In the case of protocol frames this field is normally zero-length. In the case of message transmission this field holds the message content.
Frame Stop Pattern	32-bit unsigned integer in network byte order having the hexadecimal value `55AA00FF`

The framing structure allows the recipient to find a clearly delineated frame within the TCP/IP data stream. The FoxTalk™ framing technique adds a fixed overhead of 12 bytes to every transmission. After receiving a frame an application will parse the header to determine the meaning and content of the frame.

* * * THIS PAGE LEFT INTENTIONALLY BLANK * * *

4 FoxTalk™ Header

As documented above, the framing technique delineates a FoxTalk™ Frame. Within the frame is the frame header which conveys FoxTalk™ protocol information (in a FoxTalk™ Header). Below is a table representing the FoxTalk™ header.

Element	Description
Exchange ID	16-bit unsigned integer in network byte order. This value is created by the originator and echoed by the receiver in the resultant ACK, NAK, Connect or Heartbeat response.
Frame Type	Single ASCII byte having the value A, N, M, H or C.
End-Of-Exchange Indicator	Single ASCII byte having the value Y or N.

In the next sections the individual elements are documented one at a time. The terms FoxTalk™ Header and Frame Header are synonymous.

4.1 Exchange ID

Every exchange in FoxTalk™ must receive a response from the recipient. The Exchange ID field is meant as a method to double check that the response received is actually for the last item sent. The originator of a FoxTalk™ frame should select a unique value for the Exchange ID field. He may use a pseudo-random algorithm or an incrementing integer value. There is no requirement that the value has any relation to a previously used ID other than that it should be different from the last one used. The value must a 16-bit unsigned integer in network byte order. In the case of multi-frame data messages, each frame of a single data message must use the same Exchange ID value (as the frames together comprise a single FoxTalk™ exchange).

When a receiving application sends an acknowledgement it should set the Exchange ID field of it's response to the Exchange ID value of the frame (or frames) it has just received. Please note that the Exchange ID value applies only at the protocol level and does not relate to logical message responses. In other words, suppose a client were to generate a QV transaction to NCIC. It would construct a FoxTalk™ frame to hold the inquiry message and choose an Exchange ID field. For the purpose of example, let's say it chose hex 0A17. After receiving this message, the OpenFox™ switch would respond with a FoxTalk™ Acknowledgement frame containing an Exchange ID of hex 0A17. The OpenFox™ would then switch the inquiry to NCIC. After receiving an NCIC response, OpenFox™ would build a FoxTalk™ frame to hold the NCIC response and choose an Exchange ID field. For the purpose of example, let's say it chose hex 7453. OpenFox™ would send the data message frame to the client and the client would respond with a FoxTalk™ ACK message with the Exchange ID set to hex 7453. This example demonstrates how the Exchange ID field is related to protocol exchanges rather than transaction exchanges.

4.2 Frame Type

The Frame Type field is a single ASCII byte having one of the following values:

Frame Type	Description
C	Connection message – used to negotiate session parameters.
M	Data message.
A	Positive acknowledgement.
N	Negative acknowledgement.
H	Heartbeat
K	Key negotiation message
I	Device identification message
E	Encrypted data message

4.2.1 Type C

The Connect Frame Type indicates that a connection message will occupy the frame payload. Please see section **5 – Connect Message Details**. Type C frame headers will never contain encryption fields and will always be single-frame messages (i.e. the End-Of-Exchange field must be set to 'Y').

4.2.2 Type M

The data message Type indicates that the frame payload will contain all or part of a data message. The End-Of-Exchange indicator in the FoxTalk™ header will determine whether this frame is the end of message or not. Encryption fields will be present in the Frame Header if encryption was negotiated to 'Y' (during the connect message exchange) and absent otherwise.

4.2.3 Type A

The acknowledgement Type is used to tell the other side of the connection that the last data message was safely received. The Exchange ID field in the FoxTalk™ header must match the Exchange ID of the last received data message (the data message which is being acknowledged). The Type A frame is always a single frame (the End-Of-Exchange field must be 'Y') and must never contain any frame payload data. Encryption fields must never be present on Type A frames.

4.2.4 Type N

The negative acknowledgement may be used to inform the other side of a connection that the last received data message contained errors, or was otherwise not processed successfully. FoxTalk™ leaves the decision of how to handle NAKs up to the implementer. Reasonable actions include spilling the message to an error console, or retrying up to a reasonable retry limit. If the OpenFox™ encounters protocol errors with a received data message it will generate a NAK if possible. The Frame Payload of a Type N frame should contain a printable ASCII text error message describing the error condition encountered. Type N frames must have be single frame (End-Of-Exhange must be 'Y') and must never contain encryption header fields.

4.2.5 Type H

The FoxTalk™ Heartbeat is sent by the client when a session has reached the maximum negotiated idle time. The Type H frame is always a single frame with no payload data. The OpenFox™ will immediately return a Heartbeat to the client (with the same Exchange ID as received from the client) so that the client may verify the status of the connection as well. If the OpenFox™ does not receive a heartbeat in twice the maximum idle time (i.e. it misses two consecutive heartbeats from the client) the link will be considered dead and the connection will be closed. Likewise, if the client does not receive a response to a heartbeat from OpenFox™ within the negotiated Default Timeout it should consider the link dead and go through a close-and-retry cycle.

4.2.6 Type K

The Type K message is used to negotiate a symmetric encryption key for the AES cipher. If encryption is negotiated to 'Y' during the Connect Message exchange, the server will immediately send the client a Type K message containing a server nonce. The client must chose a random client nonce and AES key, then combined with the server nonce, concatenate all three values together and encrypt this message with the 2048-bit RSA public key of the OpenFox™. This resultant message is sent to the OpenFox™ as another Type K message. Finally, OpenFox™ responds with a third Type K message that contains the encrypted client nonce echoed back so that the client knows the encryption negotiation has completed successfully.

4.2.7 Type I

The Type I message is used after the Connect Message exchange if no encryption was negotiated, and after the encryption key negotiation phase if encryption was negotiated. Not all implementations of FoxTalk™ will use the Type I message. OpenFox™ Desktop clients use this message to send the OpenFox™ server the encrypted copy of their registered license and thus perform device identification. Some implantations (for regional servers, for instance) identify devices based off of the client's IP address. Others use a logon message. When the Type I frame is used, for example in OpenFox™ Desktop, this message must be sent before regular message exchange can commence.

4.2.8 Type E

The Type E message is used to send encrypted data messages. This type may only be used on sessions that have negotiated encryption to ‘Y’. If encryption has been negotiated, than all data messages must be Type E; any Type M messages received by OpenFox™ will be NAK’d. The Type E message follows a construction where the plain text is appended with a SHA-1 hash value of itself, then encrypted using AES in CBC mode with a randomly chosen IV (Initialization Vector). The Type E frame contains the IV and the encryption output.

4.3 End-Of-Exchange Indicator

The End-Of-Exchange Indicator is a single ASCII byte, having value ‘Y’ or ‘N’, used on Type M (data message) frames to indicate whether this frame is the end of the current data message. All other Frame Types must always have End-Of-Exchange set to ‘Y’. The recipient of a data message should not send an acknowledgement to any data message frame until the End-Of-Exchange ‘Y’ frame (termed the end of message frame) is received. If an acknowledgement to a data message is not received within the negotiated Default Timeout the message sender should resend the entire message (including all End-Of-Exchange ‘N’ - non end of message - frames).

* * * THIS PAGE LEFT INTENTIONALLY BLANK * * *

5 Connect Message Details

This section of the document will provide the details of the Connect message. As documented above, after establishing a TCP session with OpenFox™ a client must send a Connect Message. The Connect Message will contain the clients preferred session attributes. The OpenFox™ will modify any of the parameters that it needs to and return the Connect Message with the final parameters. The client must understand and be able to comply with all the parameters in the returned Connect Message. If the client is unable to comply with the parameters returned by OpenFox™ it must then disconnect and notify the user.

Below is a table depicting the Connect Message:

Element	**Description**
Major Version Number	16-bit unsigned integer in network byte order containing the major version number of the FoxTalk™ protocol in use.
Minor Version Number	16-bit unsigned integer in network byte order containing the minor version number of the FoxTalk™ protocol in use.
Maximum Frame Length	32-bit unsigned integer in network byte order containing the maximum allowed frame length, send or receive.
Maximum Idle Time	16-bit unsigned integer in network byte order containing the maximum idle time in seconds. When this idle time is exceeded a heartbeat message must be sent.
Default Timeout	16-bit unsigned integer in network byte order containing the default timeout in seconds. This is the max time a client should wait for acks or heartbeat echoes, and the minimum time it should wait between connect attempts.
Use Encryption	A single ASCII character having the value ‘Y’ or ‘N’.
Object Coding Technique	A string of 3 ASCII characters having the value “NON”, “HEX” or “B64”.
Newline Sequence	A string of 4 ASCII characters having the value of “LF “, “CR “ or “CRLF”.

Each field of the connect message is discussed in detail below.

5.1 Major Version Number

This field is a 16-bit unsigned integer in network byte order that contains the major version number of the FoxTalk™ protocol in use. This field is meant to allow the OpenFox™ message switch to simultaneously communicate with multiple clients of varying version. As of this writing the only major version number is 1.

5.2 Minor Version Number

This field is a 16-bit unsigned integer in network byte order that contains the minor version number of the FoxTalk™ protocol in use. This field is meant to allow the OpenFox™ message switch to simultaneously communicate with multiple clients of varying version. This book documents FoxTalk™ minor version 1.

5.3 Maximum Frame Length

This field is a 32-bit unsigned integer in network byte order that contains the maximum allowable frame length. This field should be set to the largest frame that the client is willing to handle when the client sends its Connect Message to OpenFox™. OpenFox™ will never return a value larger than what the client has specified but may return a smaller number. The client must honor the number returned by OpenFox™. Any frames that are received containing a length larger than the negotiated maximum will be rejected by OpenFox™.

5.4 Maximum Idle Length

This field is a 16-bit unsigned integer in network byte order that contains the maximum allowable idle time before a heartbeat message must be sent. The client should leave this field set to 0 in it's connect message and should use the value returned by OpenFox™.

5.5 Default Timeout

This field is a 16-bit unsigned integer in network byte order that contains the default timeout for acks/naks and heartbeats. The client should leave this field set to 0 in it's connect message and should honor the value returned by OpenFox™. This is the maximum amount of time that OpenFox™ or the client should wait for a message acknowledgement (or nak) before retrying. It is also the maximum length of time that the client should wait for a heartbeat to be echoed by OpenFox™ before considering the link dead. It is also the minimum amount of time a client should wait after a connection close before retrying the connection.

5.6 Use Encryption

This field is a single printable ASCII character having the value 'Y' or 'N'. The client should set this field to 'Y' if it wants to send and receive encrypted messages, and 'N' if not. The OpenFox™ will return a 'Y' or 'N' to signify whether or not encryption will be used. Please note that in some cases the network layer is not encrypted and CJIS security policy dictates that all law enforcement traffic over a public line must be encrypted. In these cases, even if a client requests no encryption OpenFox™ may override the value with a 'Y'. If the client in this situation is unable to support encryption it must disconnect from OpenFox™ and notify the user that encryption is required.

5.7 Object Coding Technique

This field is a string of three printable ASCII characters. The following table shows the allowable strings and their respective meanings.

String	**Meaning**
T h “NON” e	The client does not wish to send or receive binary objects in messages. OpenFox™ will replace all binary objects with text on outbound messages to the client.
c “HEX” l	All binary objects will be present in printable hex format. This includes both objects sent to and received from OpenFox™.
i “B64” e	All binary objects will be present in Base 64 format. This includes both images sent to and received from OpenFox™.

The client should set this field to the string that reflects the way it wants to handle binary objects (such as images). OpenFox™ will always honor the choice of the client in this field and will return the identical string when it returns the connect message.

5.8 Newline Sequence

This field is a string of four printable ASCII characters that represent the way the client wishes to send and receive newline sequences in text blocks. The following table lists the possible values and their respective meanings.

String	Meaning
“LF “	New lines are demarked by a single ASCII linefeed character (hex value 0A).
“CR “	New lines are demarked by a single ASCII carriage return character (hex value 0D).
“CRLF”	New lines are demarked by an ASCII carriage return followed by an ASCII linefeed (hex value 0D0A).

Please note that in the case of “CR “ and “LF “ the trailing white space is two ASCII space characters (hex value 20). This string must always be four characters long. The client should set its preferred method of recognizing newlines in its connect message to OpenFox™. OpenFox™ will always honor the client’s choice in this field and will return the identical string when it returns the connect message to the client. All messages from the client to the OpenFox™, and from OpenFox™ to the client, will and must use the negotiated newline sequence. Any messages from the client that do not use a newline sequence matching the negotiated method may result in errors from the OpenFox™.

6 Encryption

The FoxTalk™ protocol supports FIPS 140-2 compliant encryption. In order to meet FoxTalk™ standards, a client will have to use a FIPS 140-2 validated encryption software library that supports the following cryptographic routines:

- 2048-bit Public Key RSA Encrypt with PKCS1 padding
- 160-bit SHA-1 hash
- Arbitrary Length Cryptographically Secure Psuedo-Random Number Generation
- 128-bit AES in CBC mode with PKCS7 padding

The FoxTalk™ protocol uses the RSA algorithm during Key Negotiation to protect the secret AES key and also to perform one-way authentication of the OpenFox™ server. The random number generator is used to chose session values (nonces and AES key) and also on each data message to chose an IV value. The SHA-1 hash is used to ensure the decryption results in valid plain-text (message validation). The FoxTalk™ protocol uses the Advanced Encryption Standard (AES) for the encryption of messages. AES uses the Rijndael encryption algorithm. Currently FoxTalk™ supports running AES with 128-bit keys, 128-bit blocks, CBC (Cipher Block Chaining) mode, and PKCS7 padding.

Below is a more detailed explanation of the encryption components used by FoxTalk™.

6.1 Key Negotiation

As documented above, after encryption is negotiated to 'Y' during the Connect Message exchange, the client and server must exchange a series of three Key Negotiation Messages (Type K frames) to set the AES encryption key for the session. These messages are detailed below in the proper order.

6.1.1 K1 Message

First, the server will send a Type K message to the client that contains the server nonce value (this message is nominally referred to as the K1 message). The data portion of the frame will contain only 16 bytes (128 bits) of server-chosen random data. The client must return this value, known as the server nonce, to prevent replay attacks against the server.

6.1.2 K2 Message

After receiving the K1 message from the server, the client must construct a message that returns the server nonce and gives the server a randomly chosen session AES key as well as a client nonce. This message is nominally referred to as the K2 message. The client must first use its cryptographically secure pseudo-random number generator to construct a 16 byte (128 bit) client nonce, and then a 16 byte (128 bit) AES key. It then concatenates these values together with the server nonce in sequence as follows:

128 bit client nonce	128 bit AES key	128 bit server nonce

This sequence is then run through the SHA-1 hashing algorithm which produces a 20 byte (160 bit) hash value. This hash is appended to the data, so that the sequence is now:

128 bit client nonce	128 bit AES key	128 bit server nonce	160 bit SHA-1 hash

The total length of this is 544 bits, or 68 bytes. The next step for the client is to encrypt this data to protect it on the wire for transmission to the OpenFox™ server. The algorithm used to protect this K2 message is 2048-bit RSA. The client is presented (through an out of band method) with the RSA Public Key for the OpenFox™ server (in the case of OpenFox™ Desktop, this key is built into the product). The client uses this key, with PKCS1 padding mode, to encrypt the above 68 bytes of plaintext using an RSA Public Key Encrypt method call. This call will result in 2048 bits (256 bytes) of cipher text. This cipher text is the entire data content of the K2 message which is now sent to OpenFox™.

6.1.3 K3 Message

The OpenFox™ server receives the K2 message from the client and decrypts it using its RSA private key. The decryption produces the 68 bytes of plain text constructed by the client. The server ensures that the SHA-1 hash value is correct and that the server nonce has been returned correctly, then sets the AES session key to the value chosen by the client. The final step in encryption negotiation is to let the client know the negotiation is complete by returning the client's nonce. The OpenFox™ server does this with this final Type K message, nominally referred to as the K3 message. To perform this step the encryption method uses is now identical to the method used for all Type E frames, nominally referred to as the FoxTalk™ Frame Encryption Technique.

The OpenFox™ server first computes the SHA-1 hash value of the plain text data. In this case, the K3 message has only the client nonce as the plain text. So, the server runs the 128 bit (16 byte) client nonce data through the SHA-1 hash algorithm and appends the resultant 160 bit (20 byte) output to the plain text as follows:

128 bit client nonce	160 bit SHA-1 hash value

This results in 188 bits (36 bytes) of cipher input. Next, the server chooses a random 128 bit (16 byte) Initialization Vector. This vector is used to run the above cipher input through the AES encrypt algorithm, in CBC mode, with PKCS7 padding. The AES algorithm, with padding, will result (in this case) in 48 bytes of cipher text output. The final K3 data is constructed by pre-pending the cipher text output with the IV value as follows:

128 bit Initialization Vector	48 byte encrypted K3 data

This message, now 64 byte long, is sent to the client thereby completing the encryption negotiation. The client should decrypt this K3 message, check that the hash value is correct, and then ensure that its client nonce has been returned properly. Any failure should result in an immediate disconnect from the server.

6.2 FoxTalk™ Frame Encryption Technique

This section documents the technique used by the FoxTalk™ protocol to encrypt and decrypt all Type E frames (as well as the K3 message above).

6.2.1 Encryption Technique

The encryption technique is:

- Compute the SHA-1 hash value of the plaintext
- Append the SHA-1 hash value to the plaintext to construct the AES cipher input
- Randomly choose a 128 bit (16 byte) Initialization Vector (IV)
- Use the above IV along with the AES session key to encrypt the cipher input
- Construct the frame data by first copying the IV, then the cipher output

6.2.2 Decryption Technique

The decryption technique is the exact reverse:

- Copy the first 128 bits (16 bytes) into the IV value and use it along with the AES session key to decrypt the remaining received bytes.
- Take off the last 160 bits (20 bytes) of the output and copy into a received hash value
- Compute the SHA-1 hash value of the remaining bytes and make sure it matches the received hash value

6.2.3 Frame Length Calculation

There is one more thing to be aware of when conducting encryption operations, namely, the negotiated maximum frame length. There is a calculation to make sure that the plain text input is small enough so that when the 20 byte SHA-1 hash, the PKCS7 padding, and the 16 byte IV are all added the total will still be less than the maximum frame length. To get this value, perform the following steps:

- Start with the maximum frame length value
- Subtract 12 bytes (for the start pattern, frame length, and stop pattern)
- Subtract the FoxTalk™ header length (4 bytes)
- Subtract the IV length (16 bytes)
- Round the value down to the nearest smaller number that is evenly divisible by the cipher block length (16 bytes)
- Subtract the Hash length (20 bytes)
- Subtract 1 (must reserve at least one byte for PKCS7 padding)

6.2.3.1 Example 1

To illustrate, here is an example of the above calculation done if OpenFox™ negotiated a maximum frame length of 8000:

- Start with maximum frame length: 8000
- Subtract 12 bytes (for the start pattern, frame length, and stop pattern): 7988
- Subtract the FoxTalk™ header length (4 bytes): 7984
- Subtract the IV length (16 bytes): 7968
- Round the value down to the nearest smaller number that is evenly divisible by the cipher block length (16 bytes): 7968 (already evenly divisible by 16).
- Subtract the Hash length (20 bytes): 7948
- Subtract 1 (must reserve at least one byte for PKCS7 padding): 7947

So, we would have to block plaintext into 7947 byte blocks before encrypting them to ensure the encryption overhead will not exceed the maximum negotiated frame length.

6.2.3.2 Example 2

Here is another example of the calculation this time if OpenFox™ negotiated a maximum frame length of 5000:

- Start with maximum frame length: 5000
- Subtract 12 bytes (for the start pattern, frame length, and stop pattern): 4988
- Subtract the FoxTalk™ header length (4 bytes): 4984
- Subtract the IV length (16 bytes): 4968
- Round the value down to the nearest smaller number that is evenly divisible by the cipher block length (16 bytes): 4960
- Subtract the Hash length (20 bytes): 4940
- Subtract 1 (must reserve at least one byte for PKCS7 padding): 4939

In this case, we would have to block plaintext into 4939 byte blocks before encrypting them to ensure the encryption overhead will not exceed the maximum negotiated frame length.

7 Device Identification

This section of the document discusses the identification of devices. After exchanging the Connect Message sequence, the OpenFox™ server must identify the device with which it is communicating. Although the FoxTalk™ protocol is in use in many different situations, the most common methods of device identification are:

- Regional Server/Gateway identified by peer IP address
- 3rd Party Client identified by a logon transaction
- OpenFox™ Desktop/Messenger client identified by a license transmission

This document will only consider the case of the OpenFox™ Desktop/Messenger implementation, since the others are non-standard and vary according to installation.

After finishing the encryption negotiation (or immediately after the Connect Message exchange in the case of non-encrypted sessions) the OpenFox™ Desktop client must send the local copy of its license file to the OpenFox™ server. If encryption has been negotiated on this session, then this message should be encrypted using the standard FoxTalk™ Frame Encryption Technique (detailed above in section 6.2.1) and sent with a Type of I. The OpenFox™ server will receive the Type I message (and decrypt it if required), which will now contain the local license file (this file is encrypted with a secret key that only the server knows). The server now uses its secret license key to decrypt the license file and validate that the identification is acceptable. If so, the server constructs a message back to the client to give the client its detailed runtime information including:

- The workstation's OpenFox™ Station Name
- The workstation's Default ORI
- The workstation's Agency ID
- The workstation's Agency Name
- The Local Encryption Key (used to uniquely encrypt the workstation's mail folders).

This response message is sent back to the client as another Type I frame (that is also encrypted if required). Upon receiving this message, the identification phase is complete and the client can commence regular message exchange.

* * * THIS PAGE LEFT INTENTIONALLY BLANK * * *

Appendix A – FoxTalk™ Examples

This Appendix contains several complete examples of FoxTalk™ frame exchanges to be used as a reference. The characters in the hex representations have background colors according to the part of the frame they occupy as follows:

Red:	FF00AA55	Frame Start or Stop Pattern
Magenta:	00000024	Frame Length Field
Yellow:	00014359	Frame Header Field
Blue:	4236344C	Frame Payload

Example 1: Connect message exchange

In this example a client has just established a successful TCP session with OpenFox™ and wishes to negotiate the following parameters:

FoxTalk™ Version 1.0
Max Frame Length 65,000
Use Encryption is No
Preferred Object Encoding is Base 64
Preferred new-line sequence is Linefeed only

Below is a hex representation of the ensuing frame:

```
FF00AA5500000024000143590001000000000FDE8000000004E423634
4C46202055AA00FF
```

The frame start and stop patterns are present at the each end of the frame, and the frame length is hex 24 bytes (which is decimal 36). As can be verified by counting the bytes, this represents the total size of the frame from the first FF to the last FF. The Frame Header (yellow) is deconstructed to:

Field	Content	Description
Exchange ID	0001	ID value was chosen arbitrarily by the client
Frame Type	43	ASCII 'C' for a Connect Message Type
End of Exchange Indicator	59	ASCII 'Y' to signify end of message

The Frame Payload (blue) is a Connect Message which is deconstructed to:

Field	Content	Description
Major Version Number	0001	Major version 1
Minor Version Number	0000	Minor version 0
Maximum Frame Length	0000FDE8	Hex FDE8 is decimal 65,000 which is the maximum frame length the client can handle
Maximum Idle Time	0000	Set to zero by client as per specification (section 5.4)
Default Timeout	0000	Set to zero by client as per specification (section 5.5)
Use Encryption	4E	ASCII 'N' to signify no encryption
Object Encoding Technique	423634	ASCII string of value "B64" to signify Base 64 object encoding technique
Newline Sequence	4C462020	ASCII string of four bytes value "LF " to signify use Linefeeds only for newlines.

After receiving this message the OpenFox™ switch will adjust the parameters and return a Connect Message. For the purpose of example, let's presume that OpenFox™ in this case is configured for the following parameters:

FoxTalk™ Version 1.0
Maximum Frame Length 8,000
Use Encryption – yes or no accepted
Maximum Idle Time is 3 minutes (180 seconds)
Default Timeout is 30 seconds

In this case, OpenFox™ must lower the maximum frame length requested by the client to 8,000 bytes. OpenFox™ will also present the two time fields (Max Idle Time and Default Timeout) and will honor the remaining values requested by the client. Please note that if the OpenFox™ had a maximum frame size of 120,000, that OpenFox™ would have respected the 65,000 byte maximum requested by the client.

Below is a hexadecimal representation of the ensuing response frame:

```
FF00AA5500000024000143590001000000001F4000B4001E4E423634
4C46202055AA00FF
```

Again, this frame contains the start and stop pattern and has a length of hex 24 (decimal 36). Below is a breakdown of the Frame Header:

Field	Content	Description
Exchange ID	0001	ID value chosen by the client is returned by OpenFox™
Frame Type	43	ASCII 'C' for a Connect Message
End of Exchange Indicator	59	ASCII 'Y' to signify end of message

The breakdown of the Connect Message in the Frame Payload is:

Field	Content	Description
Major Version Number	0001	Major version 1
Minor Version Number	0000	Minor version 0
Maximum Frame Length	00001F40	Hex 1F40 is decimal 8,000 which is the maximum frame length the OpenFox™ can handle. Since this was smaller that the client's value it was overridden by OpenFox™
Maximum Idle Time	00B4	Hex B4 is decimal 180 specifying a maximum idle time of 3 minutes
Default Timeout	001E	Hex 1E is decimal 30 specifying a default timeout of 30 seconds.
Use Encryption	4E	ASCII 'N' to signify no encryption
Object Encoding Technique	423634	ASCII string of value "B64" to signify Base 64 object encoding technique as requested by the client
Newline Sequence	4C462020	ASCII string of four bytes value "LF " to signify use Linefeeds only for newlines as requested by the client

After this frame is sent by OpenFox™ there is now an open FoxTalk™ session between the client and the OpenFox™. The final negotiated session parameters are:

FoxTalk™ Version 1.0
Maximum Frame Length 8,000
Maximum Idle time 3 minutes
Default Timeout 30 seconds
No encryption will be used
Objects will be encoded with the Base 64 method
All text newlines will be represented by Linefeed characters only

Example 2: Heartbeat exchange

This example will cover a case where a client's connection to OpenFox™ has been idle for the maximum allowed idle time (as negotiated with the Connect Message exchange). The client must now construct a FoxTalk™ Heartbeat Frame and deliver it to OpenFox™. Below is a hexadecimal representation of the heartbeat frame:

```
FF00AA55000000101B04485955AA00FF
```

This frame has the start and stop patterns at the beginning and ending of the frame, and has the frame length field set to hex 10 (decimal 16) which is the length of the entire frame.

Since this frame is a Heartbeat type it does not contain a payload. Also, since it is not a Data Message type it will never include the encryption fields of the FoxTalk™ header (even if encryption had been negotiated to 'Y' on this session). Below is a deconstruction of the FoxTalk™ Header:

Field	Content	Description
Exchange ID	1B04	ID value chosen arbitrarily by the client
Frame Type	48	ASCII 'H' for a Heartbeat Exchange
End of Exchange Indicator	59	ASCII 'Y' to signify end of message

When OpenFox™ receives this frame it will reset the connection's idle timer and echo the heartbeat back with the following frame (again in hexadecimal representation):

```
FF00AA55000000101B04485955AA00FF
```

This frame is identical to the frame received from the client. It is of length hex 10 (decimal 16) and contains no payload or encryption fields. Below is the breakdown for the FoxTalk™ header:

Field	Content	Description
Exchange ID	1B04	ID value chosen by the client is returned by OpenFox™
Frame Type	48	ASCII 'H' for a Heartbeat Exchange
End of Exchange Indicator	59	ASCII 'Y' to signify end of message

Example 3: Non-encrypted single frame data message

In this example the client sends a simple QV transaction to OpenFox™ on a session which has negotiated no encryption. OpenFox™ will respond with an acknowledgement. The purpose is to demonstrate single framing and non-encrypted data message exchange.

For this example, the message from the client will be an OFML QV transaction as follows:

```
<OFML>
 <HDR>
  <ID>12345ABCDE</ID>
  <DAC>SP01</DAC>
  <REF>123123123</REF>
  <MKE>QV</MKE>
  <ORI>INXML0000</ORI>
  <SUM>"QV:EXAMPLE LIC/ABC123"</SUM>
 </HDR>
 <TRN>
  <LIC>ABC123</LIC>
  <LIS>IN</LIS>
 </TRN>
</OFML>
```

Below is a hexadecimal representation of the ensuing FoxTalk™ frame sent by the client:

```
FF00AA55000000CA02174D593C4F464D4C3E3C4844523E3C49443E31
32333435414243444453C2F49443E3C4441433E535030313C2F444143
3E3C5245463E313233313233313233C2F5245463E3C4D4B453E5156
3C2F4D4B453E3C4F52493E494E584D4C303030303C2F4F52493E3C53
554D3E2251563A4558414D504C45204C49432F4142433132332223C2F
53554D3E3C2F4844523E3C54524E3E3C4C49433E4142433132333C2F
4C49433E3C4C49533E494E3C2F4C49533E3C2F54524E3E3C2F4F464D
4C3E55AA00FF
```

The frame starts and stops with the appropriate patterns and has a length field of hex CA (decimal 202) which is the entire length of the frame. Below is a deconstruction of the FoxTalk™ Header:

Field	Content	Description

Exchange ID	0217	ID value chosen arbitrarily by the client
Frame Type	4D	ASCII ‘M’ for a Data Message frame
End of Exchange Indicator	59	ASCII ‘Y’ to signify end of message

The Frame Payload contains the data of the message (in this case, a series of ASCII characters representing the OFML message text).

After receiving this frame, OpenFox™ will accept the message at the protocol level (i.e. before the payload content is parsed) with the following FoxTalk™ frame:

```
FF00AA5500000010021741595 5AA00FF
```

As always the frame starts and stops with the appropriate patterns. The frame length is hex 10 (decimal 16) which represents the overall length of the frame. The header breakdown is:

Field	**Content**	**Description**
Exchange ID	0217	ID value chosen by the client is returned by OpenFox™
Frame Type	41	ASCII ‘A’ for an Acknowledgement frame
End of Exchange Indicator	59	ASCII ‘Y’ to signify end of message

After receiving this frame the client knows the OpenFox™ has safely received the QV transaction.