Specification for FoxTalk™
TCP/IP Protocol
Version 1.1

Computer Projects of Illinois, Inc.
6416 South Cass Avenue
Westmont, IL 60559
(630) 968-0244

* * * 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. While it may seem that half-session failures are unlikely, they are actually common especially in a modern network that deploys firewall devices for security. All firewalls have an idle session timeout, which when activated does not close the connection to the peers but merely removes it from the firewalls internal tables. Two communicating applications across a firewall that idles them out will experience half-session failure. Heartbeats alleviate this concern by both keeping sessions from idling out and by allowing an unattended application to recognize a failed session and automatically re-establish connectivity it a timely fashion.

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. Please see the detailed documentation of the connect message options in section **5 – Connect Message Details**. If encryption is negotiated, then the next step is key negotiation, otherwise the session goes immediately to the identification phase.

2.3 Encryption Negotiation

If encryption is negotiated during the connect message exchange then a session key must be negotiated next. The data traffic of an encrypted FoxTalk™ session is protected through the use of the AES encryption algorithm. This algorithm is symmetric in nature and requires a key value that is known to both the server and the client. During this phase, the server requests a key value from the client providing a value known as a server nonce to be sure that the response is related to this specific request. The client returns the server nonce, along with a client nonce and a randomly chosen AES key value. This entire message is encrypted using the RSA algorithm with the public key of the FoxTalk™ server (ensuring that only the FoxTalk™ server will be able to decrypt the message). After the FoxTalk™ server determines the server nonce has been correctly return, it will apply the AES key to the session and return the client nonce encrypted with the AES session key. This allows the client to confirm that the appropriate key has been applied to the session by the server.

2.4 Client Identification

After successfully negotiating encryption if required, otherwise immediately following the connect message exchange, the client must be identified by the FoxTalk™ server. In the case of remote systems (CADs, Mobile Controllers, Regional Message Switch, etc.) this step is performed automatically by the FoxTalk™ server based on the originating IP address of the client session. In the case of OpenFox™ Desktop workstations, the Desktop software must send a proper OpenFox™ License file (obtained by the Desktop client during software registration on the Desktop Launch Website). In the case of Desktop clients, the server will Nak the License file if it is invalid or incorrect, or send a textual response containing information about the session if the License File was accepted.

2.5 Regular Messaging

Immediately after the identification 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 – fixed 4 byte length, 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	Field Presence	Description
Exchange ID	Always Present	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	Always Present	Single ASCII byte having the value A, N, M, H or C.
End-Of-Exchange Indicator	Always Present	Single ASCII byte having the value Y or N.

The length of the FoxTalk™ header is fixed at 4 bytes. 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 be 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.
K	Key message – used to negotiate a session encryption key.
I	Identity message – used to exchange client identification information
E	Encrypted data message.
M	Data message (not encrypted).
A	Positive acknowledgement.
N	Negative acknowledgement.
H	Heartbeat

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 always be single-frame messages (i.e. the End-Of-Exchange field must be set to 'Y').

4.2.2 Type K

The Key Frame Type indicates that the payload contains key negotiation data. The Type K messages are only used on sessions where encryption has been negotiated during the connect message exchange. There are a

total of 3 K frames exchanged to complete key negotiation. For further details please see section **6 – Encryption**.

4.2.3 Type I

The Identity Frame Type indicated that the payload contains information used to determine the identity of a client session. This type is only used for OpenFox™ Desktop client sessions. Type I messages are always single frames.

4.2.4 Type E

The Encrypted Data Frame Type is used to transmit application data messages that have been protected with encryption. All data messages on an encrypted FoxTalk™ session must use Type E frames (the server will Nak any Type M frames received, and will never send Type M frames on encrypted sessions). The payload of Type E message follows a specific format that is documented in section **6 – Encryption**. The End-Of-Exchange indicator in the FoxTalk™ header will determine whether this frame is the end of message or not.

4.2.5 Type M

The Data Frame 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.

4.2.6 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.

4.2.7 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 be single frame (End-Of-Exhange must be 'Y').

4.2.8 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.

4.3 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) and Type E (encrypted 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).

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. As of this writing the only minor version number is 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
"NON"	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.
"HEX"	All binary objects will be present in printable hex format. This includes both objects sent to and received from OpenFox™.
"B64"	All binary objects will be present in B64 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 the use of 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. Key management is handled through the key negotiation phase when a client connects. After the connect message exchange a series of Type K frames must be exchanged to encrypt the session. Once complete, all data must be sent as Type E frames. The Type E frame has a structured payload. It is constructed of the data to be transmitted, followed by a secure SHA-1 hash signature, and then padded out to a modulus 128-bit length with the PKCS7 algorithm. A random initialization vector is chosen and used to encrypt the above sequence of bytes with the negotiated session key. The payload of the Type E frame is then the initialization vector followed by the encryption output.

As discussed, using encryption in FoxTalk™ requires the use of the following cryptographic components:

- Random Number Generation
- RSA encryption
- PKCS7 padding
- SHA-1 hash
- AES operation

The rest of this section will discuss these components in detail, followed by descriptions of the Type K Frame exchanges required to negotiate a session key and the detailed composition of the Type E Frames used for data transmission.

6.1 Cryptographic Components

This section will describe each of the cryptographic components used by FoxTalk™. All cryptographic components used by FoxTalk™ are available for free in the OpenSSL open source package. Please note, however, that in instances where the underlying network is not encrypted itself to FIPS 140-2 standards, then OpenSSL is inadequate and a FIPS 140-2 certified implementation must be used.

6.1.1 Random Number Generation

There are several instances in FoxTalk™ where the generation of random numbers is required. Many cryptographic libraries come with random number generators which may be used for this purpose. If no random number generator is available, it is possible to construct one using the AES routine in CTR mode. This routine should be seeded strongly (preferably with true random data). An input component of the CTR mode is simply an incrementing counter. This feature in particular makes this method well suited for choosing initialization vectors (and in fact is recommended for IV generation by NIST).

6.1.2 RSA Encryption

The RSA algorithm is used by the client during the key negotiation phase. The client will only have to use this algorithm to encrypt (never decrypt). The algorithm must be invoked using the PKCS1 padding method. The key to use is the FoxTalk™ server's public key. The FoxTalk™ protocol does not govern the distribution of the server's public key. It must be obtained directly from the administrators of the FoxTalk™ server to which communication is desired. The public key is presented as a set of components that are each a string representation of a big number. The public key file contains simply the RSA 'N' (product) value and the exponent value. These values should be used to create a new instance of an RSA public key object.

6.1.3 PKCS7 Padding

Since AES is a block cipher every message must be padded out to a full block length. FoxTalk™ uses 128-bit blocks, which are 16 bytes long. The padding technique employed in PKCS7 uses the following rules:

Pad the last block to 16 bytes with characters that have a numeric value equal to the number of padding characters required.

On messages where the last block is 16 characters exactly, add a block with 16 characters each of value decimal 16 (hex 10) – which is the count of padding characters added.

This makes the recipient's job very simple; just read the last byte of the message, interpret it as a number, and remove that count of bytes from the end of the message.

Examples:

Suppose the last block of a message contains 4 bytes. The block must be padded with 12 characters, so 12 bytes of value decimal 12 (hex 0C) are appended to the message.

Suppose the last block of a message contains 11 bytes. The block must be padded with 5 characters, so 5 bytes of value decimal 5 (hex 05) are appended to the message.

Suppose the last block of a message contains 16 bytes. An entire block must be appended to the message, so 16 bytes of value decimal 16 (hex 10) are appended to the message.

6.1.4 SHA-1 Hash

The FoxTalk™ protocol requires the use of the SHA-1 hash in various places. The FoxTalk™ protocol implements the hash through best coding practice which consists of "hashing the hash" value which is recommended to alleviate security risks. Whenever a hash is called for in FoxTalk™, the data string should be run through an instance of the SHA-1 algorithm, resulting in a 160-bit (20 byte) output value. This 160-bit value should itself be run through SHA-1, producing a final (but different) 160-bit value, which is used as the FoxTalk™ hash. It is important that this "double hash" be performed or the hash values produced by the server and client will not match and the server will fail the connection.

6.1.5 AES Operation

As documented above FoxTalk™ uses AES in CBC (Cipher Block Chaining) mode with 128-bit blocks and 128-bit keys. In the CBC mode the current block of plain text is first run through a byte-at-a-time logical XOR with the previous block of cipher text before being presented to the encryption algorithm. Although many available encryption libraries will perform this operation for the caller (if he specifies CBC mode) it's certainly possible to perform the CBC processing on one's own. When the CBC mode is used there is a special condition on the very first block (since there is not a previous cipher text block). On the first block an arbitrary block is used for the XOR operation. This block is termed the Initialization Vector. A secure random number generator must be used to create each Initialization Vector (abbreviated as IV).

6.2 Key Negotiation

This section will describe the steps required to negotiate a session key. Immediately following the connect message exchange (and after negotiating that encryption will be used) the server will begin the key negotiation. The are a total of three Type K frames exchanged to negotiate the session key. The three frames are called the K1, K2 and K3 messages. An overview of the procedure is given below, followed by a detailed description of each of the three messages.

As stated above, the server will begin the key negotiation step by sending a K1 message to the client. The K1 message consists simply of a 128-bit (16 byte) server nonce. The client records the server nonce and then uses his random number generator to construct a 128-bit AES key and 128-bit client nonce. The client then constructs a K2 message by concatenating the AES key, client nonce, and server nonce together (in that order). The concatenated data is then hashed, and the hash value appended. Then the FoxTalk™ server's public RSA key is used to encrypt the above data. The output cipher text is then sent as the message payload of a Type K frame to the FoxTalk™ server, completing the K2 message. The FoxTalk™ server will use it's RSA private key to decrypt the K2 message, verify that the proper server nonce is returned, and set the AES key as the session key. Finally, the FoxTalk™ server builds a K3 message and returns it to the client. The K3 message contains only the client nonce as data, but it is encrypted with the same technique that all future Type E frames will use; the client nonce is hashed (using the technique described in section 6.1.4 above), and the hash value is appended to the nonce. This combined data is then run through a CBC AES encryption operation with an initialization vector chosen at random. The final K3 message payload consists of the initialization vector followed by the AES encrypted cipher text. The client should ensure that the client nonce value

matches what it sent in the K2 message. If so then the key negotiation has completed successfully. If not the connection should be immediately disconnected.

The following table summarizes the K Frame exchange procedure:

Message	Encryption	Direction	Content
K1	None	From Server to Client	Server Nonce
K2	RSA	From Client to Server	AES Key Client Nonce Server Nonce
K3	AES	From Server to Client	Client Nonce

The next sub sections provide a detailed description of each key negotiation message.

6.2.1 K1 Message

The K1 message is very simple, and contains only the 128-bit (16 byte) nonce value selected randomly by the server. This message is sent by the server and received by the client and represents the beginning of the key negotiation.

6.2.2 K2 Message

The K2 message contains an RSA public key encrypted message which is made from the AES session key, client nonce and server nonce. After receiving the K1 message, the client should use its random number generator to produce a 128-bit (16 byte) AES key value and a 128-bit (16 byte) client nonce value. The AES key, client nonce, and server nonce should be concatenated in this order. The hash technique described in section 6.1.4 should then be used to produce a 160-bit (20 byte) hash value out of the concatenated data. The K2 message should then be constructed as the following table represents:

Field	Length
AES Key	16 bytes
Client Nonce	16 bytes
Server Nonce	16 bytes
Hash Value (of the previous 3 data fields)	20 bytes

The client will now have a message that is 68 bytes long. These 68 bytes should be sent encrypted by calling the RSA public encrypt function with the FoxTalk™ server's RSA public key and PKCS1 padding. The resulting output (cipher text) is then sent to the server as a Type K frame, concluding the K2 message. The FoxTalk™ server uses 2048 bit RSA, which means that the resulting cipher text from the above operation will be 2048 bits long, which is 256 bytes.

6.2.3 K3 Message

After receiving the K2 message the server will use it's RSA private key to decrypt the frame payload and recover the AES key, client nonce and server nonce. If the server nonce does not match the value sent in the K1 message, the server will drop the connection. Otherwise, the server will set the session key to the AES key received from the client and then construct a K3 message. The K3 message is made out of the client nonce. The nonce value is hashed (using the technique described in section 6.1.4). The client nonce and hash value are concatenated together to form the plain text of the K3 message. The message so far is 36 bytes long (16 byte client nonce + 20 byte hash value) and is represented in the following table:

Field	Length
Client Nonce	16 bytes
Hash	20 bytes

Next, a random 128-bit (16 byte) IV (initialization vector) is chosen and the 36 bytes of plain text are padded, using the PKCS7 technique, to the next modulo 16 value (in this case 48) and then encrypted, resulting in 48 bytes of cipher text. The final frame payload is constructed of the IV plus the cipher text (a total of 64), represented in the following table:

Field	Length
Initialization Vector	16 bytes
Encrypted Data	48 bytes

To decrypt this message, the client must remove the first 16 bytes and store as the IV. Then, use the IV and the AES session key to decrypt the 48 bytes of cipher text and remove the PKCS7 padding. This will leave 36 bytes. The last 20 bytes are removed and stored as the server's hash value. The remaining 16 bytes should be hashed (using the technique described in section 6.1.4) and the resulting value checked against that sent by the server. If they match, then the client can compare these 16 bytes to the client nonce sent in the K2 message. If everything matches, then the encryption negotiation is complete. If any error should occur the connection should be closed. Please see Appendix A for examples of K messages.

6.3 Type E Frame

As mentioned previously, once an encrypted session has been negotiated all data must be transmitted as a Type E frame, never as a Type M. Type E frames are constructed with the methodology employed to create the K3 message used during key negotiation (substituting plain text for client nonce), reviewed below in section 6.3.2. Like Type M frames, Type E frames are used for data, and may need to transport data that is too large to fit in a single frame. In this case, the same multi-framing technique used for Type M frames is used for the Type E frames, with the last frame specifying the End-Of-Exchange is 'Y'.

There is one additional concern in building Type E frames that is more complex then Type M frames. Because of the overhead introduced by the encryption techniques, it is more difficult to determine the maximum amount of plaintext that may be used to construct a single Type E frame without the resultant payload size overflowing the maximum negotiated frame length. Section 6.3.1 below explains the necessary calculations.

6.3.1 Maximum Plain Text Lenth

To determine the maximum length of plain text that can be used to construct a Type E frame, we must start with the maximum negotiated frame length and work our way down. Let's name the maximum negotiated frame length MaxFL for short. First, we must remove the framing overhead (12 bytes) and the length of the FoxTalk™ header (4 bytes), resulting in (MaxFL – 16). Now we must remove the length of the Initialization Vector, which is 16 bytes, resulting in (MaxFL – 32). We now must make sure that we have a modulo 16 value, which can be found with integer math. Simply divide the number so far by 16 (which will drop the remainder) and multiple by 16, making our formula so far

(((MaxFL – 32) / 16) * 16). From this we must subtract the length of the hash value (20) and the length of at least 1 byte for padding (the PKCS7 standard will always add at least one byte - since we started with a modulo 16 value we are effectively forcing a 1 byte pad). This makes our complete formula for the maximum plain text length to be:

(((MaxFL – 32) / 16) * 16) – 21

For example, if 8000 was the maximum frame length, the maximum plain text that could be used works out to 7947. If our maximum frame length is 5000, the formula yields 4939. Remember to drop the fraction when dividing by 16. In other words, 5000 – 32 is 4968. 4968 / 16 is 310, not 310.5. This can be easily accomplished in software by using integer division (instead of floating point).

6.3.2 Type E Payload

As mentioned above, the Type E frame uses a payload structure identical to that used to compose the K3 message during key negotiation. First, the calculation in section 6.3.1 above must be used to determine a suitable length of plain text for the frame. Taking care to use a value equal to or less than that given by the formula, the plain text is first hashed using the technique in section 6.1.4. The 20 bytes of hash output are appended to the plain text. This yields a structure depicted in the following table:

Field	**Length**
Plain Text	Variable

Hash	20 bytes

This data is then encrypted, requiring that PKCS7 padding be applied and a random IV chosen, then the AES algorithm is invoked (in CBC mode using the negotiated session key) to produce cipher text output. The payload of the Type E frame is then constructed out of the IV followed by the cipher text output:

Field	**Length**
Initialization Vector	16 bytes
Cipher Text	Variable (Modulo 16)

Please see Appendix A for examples of Type E frames.

* * * 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 1.1
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:

FF00AA5500000024000143590001000100000FDE8000000004E423634
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	0001	Minor version 1
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 1.1
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:

FF00AA5500000024000143590001000100001F4000B4001E4E423634
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	0001	Minor version 1
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 1.1
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:

```
FF00AA5500000010 1B044859 55AA00FF
```

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):

```
FF00AA5500000010 1B044859 55AA00FF
```

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
3E3C5245463E313233313233313233 3C2F5245463E3C4D4B453E5156
3C2F4D4B453E3C4F52493E494E584D4C303030303C2F4F52493E3C53
554D3E2251563A4558414D504C45204C49432F414243313233223C2F
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.

Example 4: Negative acknowledgement

This example contains a hypothetical NAK message for the purpose of illustrating what a NAK would look like.

For the purposes of example, suppose OpenFox™ received a frame from the client that was an invalid K2 message. Also, suppose the frame had an Exchange ID value of 4057. The NAK message from OpenFox™ would look like:

FF00AA5500000022 40574E59 496E76616C6964204B32204D65737361676 55AA00FF

As always this frame begins with the start pattern and ends with the stop pattern. The frame length is hex 22 (decimal 34) which is the length of the entire frame. Below is a breakdown of the FoxTalk™ header:

Field	Content	Description

Exchange ID	4057	ID value chosen by the client is returned by OpenFox™
Frame Type	4E	ASCII 'N' for a Negative Acknowledgement
End of Exchange Indicator	59	ASCII 'Y' to signify end of message

The Frame Payload contains an ASCII data string of value "Invalid K2 Message". This is a textual explanation of the reason for the NAK.

Example 5: Key Negotiation

This example shows the exchange of the three K Frames to complete a key negotiation on a session for which encryption has been negotiated.

Immediately after the connection message exchange has been completed, the FoxTalk™ server sends the K1 message. Our example K1 message is:

```
FF00AA5500000020 00014B59E168F4DCFCC89F4861B91F973816CAE5
55AA00FF
```

The frame length here is hex 20, decimal 32. This is 12 bytes of framing, 4 bytes of FoxTalk™ header, and 16 bytes of server nonce. The breakdown of the header is

Field	Content	Description
Exchange ID	0001	ID value chosen by OpenFox™
Frame Type	4B	ASCII 'K' for a Key Frame
End of Exchange Indicator	59	ASCII 'Y' to signify end of message

The Client response to this frame follows:

```
FF00AA5500000110 12344B5907DBA5E832E07080960 5125C65E68F12
03A5165C5B1EC3C71EF61082A44A339DF47A86024AC4AC71B5DE6318
24A827C7A459886D30AFAA3CA8CEA3ADDC3A430E5EF59FCA34BB048A
A0990074DD4459F8DAA98BC7E03ACC92E57C39E202070560F35BA13C
3B59E7A5BB9A0EB10262E84A4DA87A78A35146CFF3485969FA701023
9B369E225958507692782A473D9A7CB033DD493CC7A6E24E6A97D6C1
D636F7D9295A7A8ABAF653F1D5A0593A911B50A3B7C734083B54D931
1DAE13CD1D4E44B70911323781BF9FBB6A14016ED1E1F4A05453EDC3
69A122E9888AE2AA3A086CE3E2E676ED84D6795F2D12518101448C23
A240515B9141B6B7168D7C6D20351064 55AA00FF
```

Again – the framing is applied, and the overall length of this frame is hex 110, decimal 272. The OpenFox™ is using 2048-bit RSA. Subtracting the overhead of 16 bytes from 272 leaves 256 bytes, which is exactly 2048-bits. Thus we can see that the

payload of this frame is the RSA encrypted output of the K2 message body, documented in section 6.2.2. Finally, the OpenFox™ responds with the K3 message:

```
FF00AA5500000050 12344B59 E672179902BBE5BEAF424EF634F92186
3D8513B429AC63D88F2157EDA3296559D19E6097AFBA57BF1C94F563
B645324E54168E8ABBBCAFED62931B408A1988A5 55AA00FF
```

As expected, the length of this frame is hex 50, decimal 80. The length of 80 minus 16 bytes of overhead leaves 64 bytes, which is the correct length for a K3 message payload, as documented in section 6.2.3.

Example 6: Encrypted Message

This example shows an encrypted message from the client to OpenFox™. The client sends the following message on a session for which encryption has been negotiated and a session key determined:

```
FF00AA5500000060 04D24559 9783822860ED6106B4C9980A93B6B2DC
63258C87932EFAC2822B2C4A1016CD17ABA75819A71EC0154B057A91
1EE4DF13760C16ECD52A81B379F0EF8EEAD29E13B6DE99CBE95B3DC4
6D3B7D0BBBC8002C 55AA00FF
```

The length of this frame is hex 60, decimal 96. Taking off the 16 bytes of overhead (12 framing, 4 header) leaves 80 bytes of payload. The first 16 of these is the IV. After removing them we are left with 64 bytes of cipher text. Please note that this value is modulo 16. The FoxTalk™ Header breakdown is:

Field	Content	Description
Exchange ID	04D2	ID value chosen by the client
Frame Type	45	ASCII ‘E’ for a Encrypted frame
End of Exchange Indicator	59	ASCII ‘Y’ to signify end of message

Upon receiving this, OpenFox™ would issue the following Ack:

```
FF00AA5500000010 04D24159 55AA00FF
```

Please note that the ID value is returned intact.