InterSystems™
IRIS Data Platform

TLS Guide

Version 2023.1
2024-04-15

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

TLS Guide
InterSystems IRIS Data Platform Version 2023.1 2024-04-15

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Tables

1

About TLS

Transport Layer Security (TLS) provides strong protection for communication between pairs of entities. It allows you to perform authentication, data integrity protection, and data encryption. It is the successor to the secure sockets layer (SSL).

SSL was created at Netscape in the mid nineteen-nineties. TLS was created as a standardization of SSL 3.0; TLS version 1.0 was released in 1999. The latest version of TLS available with InterSystems IRIS is 1.3, often known as TLS v1.3. Among the supported versions of TLS for InterSystems IRIS® data platform, InterSystems recommends the use of the latest version available.

Note: In InterSystems documentation, the terms SSL/TLS and SSL are equivalent to TLS.

A TLS connection uses a client/server model; two entities establish a TLS connection through a *TLS handshake*. When two entities complete the handshake, this means that:

- The client has authenticated the server.
- If the server requires client authentication, this has happened. (If the client and the server have both authenticated each other, this is known as mutual authentication.)
- The client and server have agreed upon session keys. (*Session keys* are the keys for use with a symmetric-key algorithm that allow the entities to protect data during subsequent communications.)
- Subsequent communication can be encrypted.
- The integrity of subsequent communication can be verified.

The *cipher suites* of the client and server specify how these activities occur as part of the handshake or are supported for a protected connection. Specifically, a peer's cipher suites specify what features and algorithms it supports. The client proposes a set of possible ciphers for use; from among those proposed, the server selects one. (If there are no common ciphers between the client and server, the handshake fails.)

To perform the handshake, TLS typically uses public-key cryptography (though it can use other means, such as the Diffie-Hellman protocol). With public-key cryptography, each peer (either the client or the server) has a public key and a private key. The private key is a sensitive secret value and the public key is a widely published value; typically, the public key is encapsulated in a certificate, which also contains identifying information about the holder, such as a name, organization, location, issuer validity, and so on. For InterSystems IRIS, a TLS *configuration* (described in About Configurations) specifies a named set of TLS-related values, including a certificate file, a private key file, and an optional set of cipher suites.

If successful, the handshake creates session keys that are used to protect subsequent communications.

While InterSystems IRIS and applications require various interactions with TLS, the end-user typically has no such direct interactions. For example, a browser uses TLS to establish a secure connection with a specified web site by requiring that the site (the server, in this case) authenticate itself to the browser (which occurs unbeknownst to the browser's user) and the lock icon that appears in the browser is designed to indicate that TLS is protecting the connection.

1.1 InterSystems IRIS Support for TLS

InterSystems IRIS supports TLS to secure several types of connections:

- From various client applications that interact with the InterSystems IRIS superserver (including ODBC, JDBC, and Studio).
- From Telnet clients that interact with the Telnet server.
- For use with TCP connections where an InterSystems IRIS instance is the client or server (or an InterSystems IRIS instance is at each end).
- With the Enterprise Cache Protocol (ECP). For information on using TLS with ECP, see Securing Application Server Connections to a Data Server with TLS.

As a server, InterSystems IRIS accepts connections and establishes the use of TLS; as a client, InterSystems IRIS is able to connect to servers that require the use of TLS. In all cases, InterSystems IRIS uses what is called a TLS *configuration*, which specifies the various characteristics of an InterSystems IRIS instance as part of an TLS connection.

1.2 Which TLS Versions Does My Instance of InterSystems IRIS Support?

The versions of TLS that are available for an InterSystems IRIS instance depend on several factors:

1. The major version of the OpenSSL libraries available for the operating system (OS) version. These libraries determine the possible versions of the TLS protocol that the operating system supports.
2. Any further restrictions that the operating system vendor has placed on supported versions of the protocol, such as those on Ubuntu 20.04.
3. The minimum supported version of TLS for this version of InterSystems IRIS. For this release, it is TLS v1.0.

For containers, the supported versions of TLS depend on the operating system and version of the container host.

Important: Because the protocols vary by operating system version, two instances of the same version of InterSystems IRIS may not support the same versions of the TLS protocol. Note that TLSv1.2 is the only version of the protocol that is supported on all platforms.

OS	Version	OpenSSL Version	TLS Version	Notes
AIX	7.2	1.0.2	1.0, 1.1, 1.2	See AIX 7.2 TLS Notes below.
AIX	7.3	1.0.2	1.0, 1.1, 1.2	
Red Hat Linux	8	1.1.1	1.0, 1.1, 1.2, 1.3	See Red Hat Linux 8 TLS Notes below.
Red Hat Linux	9	3.0	1.2, 1.3	
SUSE Linux	All	1.1.1	1.0, 1.1, 1.2, 1.3	
Ubuntu Linux	18.04	1.1.1	1.0, 1.1, 1.2, 1.3	
Ubuntu Linux	20.04	1.1.1	1.2, 1.3	See Ubuntu Linux 20.04 and 22.04 TLS Notes below.
Ubuntu Linux	22.04	3.0	1.2, 1.3	See Ubuntu Linux 20.04 and 22.04 TLS Notes below.
Windows	All	1.1.1	1.0, 1.1, 1.2, 1.3	See Windows TLS Notes below.

Note: For information on versions of Oracle Linux, see the analogous version of Red Hat Linux.

1.2.1 AIX 7.2 TLS Notes

Because AIX 7.2 uses the OpenSSL 1.0.2 libraries, note that:

- The OpenSSL 1.0.2 libraries support SSLv3 through TLSv1.2. Because InterSystems IRIS does not support SSLv3, there is support only for TLSv1.0 through TLSv1.2.
- The OpenSSL 1.0.2 libraries do not include SHA-3, so InterSystems provides its own implementation of SHA-3. This implementation is not compatible with the **RSASHA3Sign** and **RSASHA3Verify** functions. Calls to these functions return an <UNIMPLEMENTED> error.

1.2.2 Red Hat Linux 8 TLS Notes

In FIPS mode, Red Hat Linux 8 supports only TLSv1.2 and TLSv1.3.

1.2.3 Ubuntu Linux 20.04 and 22.04 TLS Notes

Ubuntu 20.04 and 22.04 support only TLSv1.2 and TLSv1.3. This is because these versions of Ubuntu prohibit the use of TLSv1.0 and TLSv1.1.

1.2.4 Windows TLS Notes

Windows does not use OpenSSL, so InterSystems ships the OpenSSL 1.1.1 libraries as part of the InterSystems IRIS distribution. Hence, there is support for TLSv1.0 through TLSv1.3.

2

About Configurations

InterSystems IRIS® data platform can support multiple *configurations*, each of which specifies a named set of TLS-related values. All its existing configurations are activated at startup. When you create a new configuration from the Management Portal, InterSystems IRIS activates it when you save it. The page for managing TLS configurations is the **SSL/TLS Configurations** page (**System Administration** > **Security** > **SSL/TLS Configurations**).

2.1 Create or Edit a TLS Configuration

The page for creating or editing a TLS configuration is the **SSL/TLS Configurations** page (**System Administration** > **Security** > **SSL/TLS Configurations**). To create a new configuration, click **Create New Configuration**, which displays the **New SSL/TLS Configuration** page; to edit an existing configuration, click **Edit** to the right of the name of the configuration. (You can also create a new set of configurations for a mirror member by clicking **Create Configurations for Mirror**; for more information on mirroring and TLS, see "Configuring InterSystems IRIS to Use TLS with Mirroring.")

When creating or editing a TLS configuration, the following fields are available:

- **Configuration Name** — The string by which the configuration is identified. Configuration names can contain any alphanumeric character and any punctuation except the "`|`" character. If you are creating a configuration for the InterSystems IRIS superserver, the configuration name must be `%SuperServer`; for more information about this topic, see "Configuring the InterSystems IRIS Superserver to use TLS."
- **Description** — Any text.
- **Enabled** — Whether or not the configuration is available for activation.
- **Type** — Intended purpose for this configuration, where the choice is **Client** or **Server**; the default is **Client**. Clients initiate the use of the protocol and servers respond to the initial request. (The InterSystems IRIS superserver uses a server configuration; TLS clients use a client configuration.) The value chosen for this field determines:
  - Whether the next field is the **Server certificate verification** or **Client certificate verification** field. If the configuration is for a client, the next field is the **Server certificate verification**, which specifies the verification that may be required for the certificate of any server to which the client is connecting; if the configuration is for a server, the next field is the **Client certificate verification**, which specifies the verification that may be required for the certificate of any client that attempts to connect to the server.
  - The behavior of the **File containing trusted Certificate Authority certificate(s)** field.
- **Server certificate verification** or **Client certificate verification** — Specifies whether or not the configuration requires the verification of the certificate of the peer to which it is connecting.

  A configuration for a client must have a specified **Server certificate verification** and supports possible values of:

  - **None** — Continues under all circumstances.
  - **Require** — Continues only if certificate verification succeeds.

  A configuration for a server must have a specified **Client certificate verification** and supports possible values of:

  - **None** — Specifies that the server neither requests nor requires a client certificate.
  - **Request** — Allows the client to provide or not provide a certificate. If the client provides no certificate, then authentication proceeds; if the client provides a certificate and verification fails, then authentication fails.
  - **Require** — Specifies that the client must provide a certificate; authentication depends on verification of the certificate.

- **File containing trusted Certificate Authority certificate(s)** — The path and name of a file that contains the X.509 certificate(s) in PEM format of the Certificate Authority (CA) or Certificate Authorities that this configuration trusts. The configuration uses the certificates of the trusted CA(s) to verify peer certificates. Typically, a production system uses certificates from commercial CAs with publicly available certificates.

  Regarding this field, note the following:

  - You can specify the path of the file as either an absolute path or as a path relative to the *<install-dir>*`/mgr/` directory.
  - On Windows and macOS, you can specify that the configuration uses the list of trusted CA certificates that the local operating system provides. To do so, specify the string `%OSCertificateStore` as the value of this field.

    On Windows, InterSystems IRIS is compatible with the Microsoft Root Certificate Program that uses Windows Update to fetch additional certificates on demand. For more information about how to configure certificate updating, see Configure Trusted Roots and Disallowed Certificates on the Microsoft website.
  - For a server configuration with a **Client certificate verification** value of **None**, this field is not available, since there is no peer verification.
  - Certificates from the Windows Certificate Export Wizard must be in PEM-encoded X.509 format, not the default of DER-encoded binary X.509.
  - With mirroring, the configuration must also have enough information to verify its own certificate.

  For information on how these certificates are used, see Establishing the Required Certificate Chain. For information on file names for these certificates and how to verify a certificate chain, see the OpenSSL documentation on the **verify** command.

- **This client’s credentials** or **This server’s credentials** — The files (if needed) containing the X.509 certificate and private key for the local configuration:

  - **File containing this client’s certificate** or **File containing this server’s certificate** — The full location of the configuration’s own X.509 certificate(s). This must be PEM encoded and can be specified as either an absolute or a relative path. This can include a certificate chain. For information on how this is used for authentication, see Establishing the Required Certificate Chain. (Note that certificates from the Windows Certificate Export Wizard must be in PEM-encoded X.509 format, not the default of DER encoded binary X.509.)
  - **File containing associated private key** — The full location of the configuration’s private key file, specified as either an absolute or relative path.
  - **Private key type** — The algorithm used to generate the private key, where valid options are **DSA** (Digital Signature Algorithm) and **RSA** (Rivest, Shamir, and Adleman, for the algorithm’s inventors).
  - **Private key password** — An optional password for encrypting and decrypting the configuration’s private key.

> **Note:** If the private key is password-protected and you do not enter a value here, InterSystems IRIS cannot confirm that the private key and the certificate’s public key match each other; this can result in mismatched keys being saved as a key pair.

  - **Private key password (confirm)** — A retyping of the optional password to ensure that it is the intended string.

- **Cryptographic settings**:
  - **Minimum Protocol Version** — The earliest version of the TLS protocol that this configuration supports. This is a drop-down menu that lists all the versions that the instance can support and is TLS v1.2 by default. See note below.
  - **Maximum Protocol Version** — The most recent version of the TLS protocol that this configuration supports. This is a drop-down menu that lists all the versions that the instance can support and is TLS v1.3 by default. See note below.
  - **Enabled cipherlist (TLSv1.2 and below)** — The set of ciphers used to protect communications between the client and the server, if you are using TLS v1.2 or an earlier version. See Supported Ciphers Syntax for more information on this topic.
  - **Enabled ciphersuites (TLSv1.3)** — The set of ciphers used to protect communications between the client and the server, if you are using TLS v1.3. See Supported Ciphers Syntax for more information on this topic.
  - **Diffie Hellman Bits** — (Servers only) The size in bits of the key that the Diffie Hellman ciphers use. The minimum key size varies by operating system. The **Auto** option specifies a key size that is at least the minimum required for the local operating system. Note that the Open Web Application Security Project (OWASP) recommends a minimum key size of 2048 bits.

> **Note:** As described in Which TLS Versions Does My Instance of InterSystems IRIS Support?, the versions of TLS that may be available depend on the version of the underlying OpenSSL libraries that are in use. InterSystems IRIS checks what OpenSSL libraries are in use and attempts to present only the relevant available versions of TLS. A system may, however, be configured in ways that InterSystems IRIS cannot account for. For example, Ubuntu 20.04 ships with a configuration of OpenSSL that has been modified from the default to disallow TLS 1.0 and TLS 1.1. In cases like this, InterSystems IRIS error message and logging are designed to help you determine if there is a conflict between the OpenSSL configuration and the InterSystems IRIS configuration; contact the InterSystems Worldwide Response Center (WRC) or your operating system vendor if you encounter unexpected behavior.

- **OCSP Settings**:
  - **OCSP Stapling** — Whether or not the configuration supports OCSP stapling.

    If you enable OCSP stapling for a client, then the client requests it; if the server does not provide a stapled OCSP response or the response fails validation, then the handshake fails. If OCSP stapling is enabled for a server, then the server provides a stapled OCSP response when it receives a request from a client.

    InterSystems IRIS creates the OCSP response file if it doesn't exist when you save the TLS configuration. An expired OCSP response automatically updates when you save the TLS configuration or when the server receives a request. Additionally, the CertCheck System Sensor (%SYS.Monitor.SystemSensors::**CertCheck()**) updates the response periodically. IRIS background jobs and any jobs that initiate a TLS server connection must have read/write permission on the OCSP response file. You can grant read/write access to the InterSystems IRIS effective group to satisfy these requirements.

    Regardless of whether you enable OCSP stapling, any user who can create a server-side TLS socket can do so with any enabled TLS configuration.

Note: The required fields vary, depending on whether the configuration is to be a client or server and on the desired features. Not all fields are required for all TLS configurations.

To complete the process of creating or editing a configuration, use the following buttons, which appear at the top of this page:

- **Save** — Dismisses the dialog, saving and then activating the configuration. This saves changes to an existing configuration or the configuration being created.
- **Cancel** — Dismisses the dialog without saving changes to an existing configuration or without saving a configuration being created.
- **Test** — Checks for valid configuration information. If the configuration's role is as a client, selecting this button also prompts for a server (its host name, not its URL) and a port number; InterSystems IRIS then tries to establish a test connection to that server. (This button is not available when creating a server configuration.)

  Note: The **Test** button may not be able to successfully connect with all TLS servers, even if the configuration has no errors. This is because the connection test performs a TLS handshake followed by an HTTP request. If the server expects a StartTLS message before the handshake (such as for use with LDAP, SMTP, FTPS, or another protocol), then the test fails, even though the actual TLS connection to the server succeeds.

2.1.1 Required Information for Certificates

When a client authenticates a server, the client needs to have the full certificate chain from the server's own certificate to the server's trusted CA certificate — including all intermediaries between the two.

There is an issue when setting up a server TLS configuration and the server's trusted CA certificate is not a root certificate. In order for authentication to work properly, the client needs to have access to all the certificates that constitute the certificate chain from the server's personal certificate to a self-signed trusted CA certificate. This chain can be obtained from the combination of the server's certificate file (sent during the handshake) and the client's trusted CA certificate file. The self-signed trusted root CA certificate must be in the client's CA certificate file, and the server's personal certificate must be the first entry in the server's certificate file. Other certificates may be divided between the two locations. The same constraints apply in reverse when a client authenticates to a server.

Regarding certificate formats, note that certificates from the Windows Certificate Export Wizard must be in PEM-encoded X.509 format, not the default of DER encoded binary X.509. All certificates must be PEM encoded regardless of file extension.

2.1.2 Enabled Cipher Suites Syntax

A configuration only allows connections that use its enabled cipher suites. To specify enabled cipher suites, you can either:

- Provide a list of individual cipher suites, using each one's name
- Use OpenSSL syntax to specify which cipher suites to enable and disable

Both the list of cipher suite names and the syntax for specifying enabled cipher suites is described on the ciphers(1) man page at openssl.org. This syntax allows you to specify guidelines for requiring or proscribing the use of various features and algorithms for a configuration.

The default set of cipher suites for an InterSystems IRIS configuration is `ALL:!aNULL:!eNULL:!EXP:!SSLv2` which breaks down into the following group of colon-separated statements:

- `ALL` — Includes all cipher suites except the eNULL ciphers
- `!aNULL` — Excludes ciphers that do not offer authentication

- `!eNULL` — Excludes ciphers that do not offer encryption
- `!EXP` — Excludes export-approved algorithms (both 40- and 56-bit)
- `!SSLv2` — Excludes SSL v2.0 cipher suites

For more information, see the OpenSSL documentation on the ciphers command.

2.1.3 A Note on InterSystems IRIS Client Applications Using TLS

For certain activities, you can use InterSystems IRIS instances to support client applications that interact with the InterSystems IRIS superserver.

When using client applications that interact with the InterSystems IRIS superserver using TLS, the following aspects of the configuration require particular attention:

- **Configuration Name** — While there are no constraints on the name of clients, this information is required to configure the connection.
- **Type** — Because the instance is serving with a TLS client, the type must be specified to be of type **Client**.
- **Ciphersuites** — The specified cipher suites need to match those required or specified by the server.

It is also necessary to ensure that the client and the server are configured so that each may verify the other’s certificate chain, as described in Establishing the Required Certificate Chain.

2.2 Delete a Configuration

The page for deleting a TLS configuration is the **SSL/TLS Configurations** page (**System Administration** > **Security** > **SSL/TLS Configurations**). To delete a configuration, click **Delete** to the right of the name of the configuration. The Portal prompts you to confirm the action.

2.3 Update Certificates for Existing Configurations

InterSystems IRIS provides a way for updating the associated certificates for a TLS configuration. After obtaining the new certificates, you can update them in the TLS configuration. The changes take effect with a new network connection. Many connections are dropped and reestablished quickly so the change can be immediate. However, certain connections can remain open for longer, for example, mirroring and certain interoperability interface connections. To ensure that the updated certificates take effect, InterSystems recommends you drop and reestablish these connections.

Mirroring environments require an additional step after reestablishing connections. Please see Authorizing X.509 DN Updates for more details.

2.4 Reserved and Required Configuration Names

InterSystems IRIS reserves several TLS configuration names for use with particular features. When using such a feature, you must use the reserved configuration name(s). The reserved configuration names are:

- `%MirrorClient` — For a mirror member when acting as a TLS client. For more information on mirroring and TLS, see "Configuring InterSystems IRIS to Use TLS with Mirroring."
- `%MirrorServer` — For a mirror member when acting as a TLS server. For more information on mirroring and TLS, see "Configuring InterSystems IRIS to Use TLS with Mirroring."
- `%SuperServer` — For the InterSystems IRIS superserver when accepting connections from other InterSystems IRIS components. For more information about configuring the superserver to use TLS, see Configuring the InterSystems IRIS Superserver to Use TLS.
- `%TELNET/SSL` — For the Windows Telnet server when accepting connections protected by TLS. For more information on mirroring and Telnet, see "Configuring the InterSystems IRIS Telnet Server for TLS."

Important: For TLS to function properly, you must use the exact case for each configuration name as it appears here.

2.5 Creating, Editing, and Deleting TLS Configurations Programmatically

To manage TLS configurations programmatically, use:

- The Security.SSLConfigs class
- The applicable configuration merge actions:
  - CreateSSLConfigs
  - ModifySSLConfigs
  - DeleteSSLConfigs

3

Configuring the InterSystems IRIS Superserver to Use TLS

To use TLS for communications among components of InterSystems IRIS® data platform, configure the InterSystems IRIS superserver to use TLS. To do this, the procedure is:

1. From the Management Portal home page, go to the **SSL/TLS Configurations** page (**System Administration** > **Security** > **SSL/TLS Configurations**).
2. On the **SSL/TLS Configurations** page, select **Create New Configuration**, which displays the **New SSL/TLS Configuration** page.
3. On the **New SSL/TLS Configuration** page, create a TLS server configuration with a configuration name of `%SuperServer` (using the exact case as specified here). For details about creating a TLS configuration, see Create or Edit a TLS Configuration.
4. On the **System-wide Security Parameters** page (**System Administration** > **Security** > **System Security** > **System-wide Security Parameters**), for the **Superserver SSL/TLS Support** field, choose **Enabled**. This specifies that the superserver supports (but does not require) TLS-secured connections.

   Note: If you wish to configure the superserver to require TLS-secured connections, first specify that TLS is simply enabled.

5. Set up clients to use TLS as appropriate (see Configuring InterSystems IRIS Telnet to Use TLS).

4

Configuring InterSystems IRIS Telnet to Use TLS

InterSystems IRIS® data platform offers several options for using TLS-protected Telnet connections.

4.1 Configure the InterSystems IRIS Telnet Server to use TLS

You can configure InterSystems IRIS to accept TLS-protected connections from Telnet clients. To do this, configure the InterSystems IRIS Telnet server to use TLS:

1. From the Management Portal home page, go to the **SSL/TLS Configurations** page (**System Administration > Security > SSL/TLS Configurations**).
2. On the **SSL/TLS Configurations** page, select **Create New Configuration**, which displays the **New SSL/TLS Configuration** page. On this page, create a TLS Server configuration with a configuration name of `%TELNET/SSL`.
3. Enable the Telnet service, **`%Service_Telnet`**:
   a. On the **Services** page (**System Administration > Security > Services**), click **%Service_Telnet** to display the **Edit Service** page for the Telnet service.
   b. On the **Edit Service** page, check **Service Enabled** if it is not already checked.
   c. Click **Save**.
4. On the **System-wide Security Parameters** page (**System Administration > Security > System Security**), select **Enabled** for both the **Superserver SSL/TLS support** and the **Telnet server SSL/TLS support** settings.

4.2 Configuring Telnet Clients to Use TLS

InterSystems IRIS accepts TLS connections from both the InterSystems Telnet client and third-party Telnet clients.

4.2.1 Configure the InterSystems Telnet Client to Use TLS

You can configure the InterSystems Telnet client to use a TLS connection. The process involves several steps:

1. On the instance that is the Telnet server, configure it according to the instructions in the previous section, which includes the option of requiring TLS.
2. On the instance that is the Telnet client, configure the settings file according to the instructions in "Connecting from a Windows Client Using a Settings File."

4.2.2 Configure Third-Party Telnet Clients to Use TLS

You can configure third-party Telnet clients to connect to an InterSystems Telnet server. The required or recommended configuration actions depend on the software in use and the selected cipher suites. The following guidelines apply:

- If the Telnet client requires server authentication, then the server must provide a certificate and the client must have access to the server's certificate chain.
- If the InterSystems IRIS Telnet server *requires* client authentication, then the client must provide a certificate and the server must have access to the client's certificate chain.
- If the InterSystems IRIS Telnet server *requests* client authentication, then the client has the option of providing a certificate and a certificate chain to its certificate authority (CA). If the client does not provide a certificate, then authentication succeeds; if it provides a non-valid certificate or certificate chain, then authentication fails.

For information on how certificate and certificate chains are used for authentication, see Establishing the Required Certificate Chain.

5

Configuring Java Clients to Use TLS with InterSystems IRIS

You can configure a Java client application to use TLS when it communicates with InterSystems IRIS® data platform. This communication occurs through the superserver, so a related required step is setting up the superserver to use TLS; this is described in Configuring the InterSystems IRIS Superserver to Use TLS. Java clients can be implemented using either JDBC or object bindings.

The process for configuring a Java client application to use TLS with InterSystems IRIS is:

1. Determine if the client requires a keystore or a truststore. This depends on several factors: whether or not the InterSystems IRIS server requests or requires client authentication; whether server authentication is required; and the cipher suites in use. See "Determine the Need for a Keystore and a Truststore" for more information.
2. Create a configuration file with properties in it to provide those features. See "Create a Client Configuration" for more information.
3. In the code for the client application, optionally specify the name of the client configuration; if you do not specify a name, Java uses the default configuration information. See "Specify the Use of the Client Configuration" for more information.

5.1 Determine the Need for a Keystore and a Truststore

A keystore serves as a repository for the client's private key, public key certificate, and any Certificate Authority (CA) information. This information is needed (1) if the InterSystems IRIS server requires client authentication or (2) if the cipher suite in use requires a client key pair:

- Whether or not the InterSystems IRIS server requires client authentication is determined by the choice for the **Peer certificate verification level** field on the **Edit SSL/TLS Configuration** page for that InterSystems IRIS instance's "%SuperServer" TLS configuration. If the field has a value of `Require`, the client must have a certificate; if the field has a value of `Request`, the server checks a certificate if one is available.
- The client and server agree upon a cipher suite to use. This cipher suite determines whether or not there is a client certificate, a key pair, or both. The enabled server cipher suites are determined by the value of the **Enabled ciphersuites** field on the **Edit SSL/TLS Configuration** page for that InterSystems IRIS instance's "%SuperServer" TLS configuration. The cipher suites available to the client depend on the version of Java it is using.

If the client has a private key and certificate, these are stored in the client's keystore; the keystore can also hold the client's root CA certificate and any intermediate CA certificates. To authenticate the server, the client may need to have the root CA certificate for the server and any intermediate CA certificates, these can be stored either in the client's truststore or along with client certificate information in the keystore. For more information on keystores and truststores, see the section "Keystores and Truststores" in the *Java Secure Socket Extension (JSSE) Reference Guide*.

5.2 Create a Client Configuration

The behavior of a Java client depends on the values of properties in its configuration. The configuration gets these values from what is known as a "configuration file," either from the configuration file's default values or from its configuration-specific values. The following sections describe how configuration files work:

- Configuration Files, Configurations, Properties, Values, and Defaults
- Java Client Configuration Properties
- A Sample Configuration File
- Name the Configuration File

5.2.1 Configuration Files, Configurations, Properties, Values, and Defaults

Each configuration file specifies values for the properties that one or more configurations use. The file includes both default values and configuration-specific values, in the form of name-value pairs. Generally, unversioned property names specify default values for properties and versioned property names specify configuration-specific values.

If a configuration file contains only one configuration definition, that single configuration can use unversioned properties. However, it cannot have an associated name property. Without a named configuration, invoke the configuration without specifying a name (as described in "Specify the Use of the Client Configuration" and "Specify a Configuration without a Name").

If a configuration file contains multiple configurations, each configuration is defined by the existence of a numbered version of the *name* property of the form *name.n*, where *n* is the number of the configuration. The names of a configuration's other properties use the same version number as the *name* property, so that they have a form of *propertyname.n* where *propertyname* is the name of the property and *n* is the number of the configuration.

The definitions in a configuration file are case-sensitive. Their use of spaces is flexible. The order of property definitions is also flexible.

To specify the default value of a property for use by all configurations, specify an unversioned property name and its value in the following form:

```
propertyName = propertyValue
```

For example, to specify the default value for the keyStoreType property as pkcs12, the form is:

```
keyStoreType = pkcs12
```

To override the default value for a property, specify a versioned property name, such as:

```
keyStoreType.1 = jceks
```

If a configuration file contains multiple configuration definitions, then these versions must use sequential ordering; if client application code refers to a configuration that follows a sequential gap, then an error results. For example, suppose that a configuration file has three versioned name properties: name.1, name.2, and name.4; the configuration associated with the name.4 property will not ever be created and a reference to it will fail with an error.

5.2.2 Java Client Configuration Properties

The properties are:

- cipherSuites — A comma-delimited list of supported cipher suites. The available cipher suites depend on the JRE (Java Runtime Environment) on the machine. During the TLS handshake, the server selects the strongest cipher suite that both it and the client support. [Optional]
- debug — Whether or not debugging information is logged to the Java system.err file. This property can have a value of `true` or `false` (`false` is the default). The setting of this property has no effect on exception handling. [Optional]
- keyRecoveryPassword — Password used to access the client's private key; this was created at the same time as the private key pair. [Required if the private key has password protections and application code is not passing in the private key as an input parameter.]
- keyStore — The file for storing the client private key and certificate information. The keystore can also hold the content typically associated with the truststore. [Optional]
- keyStorePassword — Password to gain access to the keystore. [Required if a password was specified when the keystore was created.]
- keyStoreType — The format of the keystore file, if one is specified. [Optional]

  Supported formats are:

  - `jks` — Java KeyStore, the Java proprietary format. [Default]
  - `jceks` — Java Cryptography Extension KeyStore format.
  - `pkcs12` — Public Key Certificate Standard #12 format.

- logFile — The file in which Java records errors. [Optional]
- name — A versioned identifier for the Java client configuration. (Each name property must be versioned. Any unversioned name property is not meaningful and is ignored.) [Optional]

  If the configuration file specifies only a single configuration and only uses unversioned property names, the name property is not required (as described in "Specify the Use of the Client Configuration"). For information about specifying multiple configurations with a single configuration file, see Configuration Files, Configurations, Properties, Values, and Defaults).

- protocol — The version of the TLS protocol to be used for the connection. [Required]

  Supported values include:

  - `TLS` — Any version of the TLS protocol. During the TLS handshake, the server selects the latest (most recent) version of the protocol that it supports. [Default]
  - `TLSv1` — Version 1 of TLS.
  - `TLSv1.1` — Version 1.1 of TLS.
  - `TLSv1.2` — Version 1.2 of TLS.
  - `TLSv1.3` — Version 1.3 of TLS.

- serverHostNameVerification – Whether or not the connection performs server hostname verification to prevent man-in-the-middle attacks. This property can have a value of true or false (false is the default). [Optional]

- trustStore — The file for storing the server's root CA certificate; it can also hold the certificates for any intermediate CAs. (This information can also be placed in the keystore.) [Optional]
- trustStorePassword — Password to gain access to the truststore. [Required if a password was specified when the keystore was created.]
- trustStoreType — The format of the truststore file, if one is specified. [Optional]

  Supported formats are:

  - `jks` — Java KeyStore, the Java proprietary format. [Default]
  - `jceks` — Java Cryptography Extension KeyStore format.
  - `pkcs12` — Public Key Certificate Standard #12 format.

5.2.3 A Sample Configuration File

The following is a sample configuration file for use with a Java client:

```
debug = false
logFile = javatls.log
protocol = TLSv1.3
cipherSuites = TLS_AES_256_GCM_SHA384
keyStoreType = JKS
keyStore = keystore.jks
keyRecoveryPassword = <password>
keyStorePassword = <password>
trustStoreType = JKS
trustStore = truststore.jks
trustStorePassword = <password>
trustStoreRecoveryPassword = <password>

name.1 = IRISJavaClient1
keyStorePassword.1 = <password>
keyRecoveryPassword.1 = <password>
trustStorePassword.1 = <password>
trustStoreRecoveryPassword.1 = <password>

name.2 = IRISJavaClient2
protocol.2 = TLS
keyStoreType.2 = pkcs12
keyStore.2 = keystore.p12
keyStorePassword.2 = <password>
trustStore.2 = cjc1.ts
trustStorePassword.2 = <password>

name.3 = IRISJavaClient3
protocol.3 = TLSv1.2
debug.3 = true
cipherSuites.3 = TLS_RSA_WITH_AES_128_CBC_SHA
```

5.2.4 Name the Configuration File

Either save the configuration file with the name SSLConfig.properties or set the value of the Java environment variable *com.intersystems.SSLConfigFile* to the name of the file. The code checks for the file in the current working directory.

5.3 Specify the Use of the Client Configuration

Once a configuration has been defined, client application code invokes it when connecting to the server. You can do this either with calls for the DriverManager object or the IRISDataSource object.

5.3.1 Use the DriverManager Object

With DriverManager, this involves the following steps:

1. Creating a Java Properties object.
2. Setting the value for various properties of that object.
3. Passing that object to Java Connection object for the connection from the client to the InterSystems IRIS server.

To specify information for the connection, the first part of the process is to create a Properties object from a configuration file and set the values of particular properties in it. In its simplest form, the code to do this is:

```
java.util.Properties prop = new java.util.Properties();
prop.put("connection security level", "10");
prop.put("SSL configuration name",configName);
prop.put("key recovery password",keyPassword);
```

where

- The connection security level of 10 specifies that the client is attempting use TLS to protect the connection.
- *configName* is a variable whose value holds the name of Java client configuration. If the configuration file has only default values and these are being used for a single configuration, do not include this line; see Specify a Configuration without a Name for details.
- *keyPassword* is the password required to extract the client's private key from the keystore.

Once the Properties object exists and has been populated, the final step is to pass it to the connection from the InterSystems IRIS Java client to the InterSystems IRIS server. This is done in the call to the **DriverManager.getConnection** method. The form of this call is:

```
Connection conn = DriverManager.getConnection(IRISServerAddress, prop);
```

where *IRISServerAddress* is a string that specifies the address of the InterSystems IRIS server and *prop* is the properties object being passed to that string.

If this call succeeds, the TLS-protected connection has been established. Typically, application code containing calls such as those described in this section includes various checks for success and protection against any errors. For more details about using InterSystems IRIS Java connectivity, see *Using Java JDBC with InterSystems IRIS*.

5.3.2 Use the IRISDataSourceObject

With the IRISDataSource object, the procedure is to create the object, call its methods to set the relevant values, and establish the connection. The methods are:

- **setConnectionSecurityLevel** — This method takes a single argument: a connection security level of 10, which specifies that the client is attempting use TLS to protect the connection.
- **setSSLConfigurationName** — This method takes a single argument: a variable whose value holds the name of Java client configuration. If the configuration file has only default values and these are being used for a single configuration, do not include this line; see Specify a Configuration without a Name for details.
- **setKeyRecoveryPassword** — This method takes a single argument: the password required to extract the client's private key from the keystore.

In its simplest form, the code to do this is:

```
try{
   IRISDataSource ds = new IRISDataSource();

   ds.setURL("jdbc:IRIS://127.0.0.1:1972/TESTNAMESPACE");
   ds.setConnectionSecurityLevel(10);
   ds.setSSLConfigurationName(configName);
   ds.setKeyRecoveryPassword(keyPassword);

   Connection dbconnection = ds.getConnection();
}
```

For a complete list of the methods for getting and setting properties, see the JDBC Quick Reference. The JavaDoc for com.intersystems.jdbc.IRISDataSource is under <install-dir>/dev/java/doc/index.html

5.3.3 Specify a Configuration without a Name

If a configuration file contains only one configuration definition, that single configuration can use unversioned properties. However, it cannot have an associated name property.

When working with a DriverManager object, the Properties object uses only the default values from the configuration file. The code for creating this object differs from the typical case in that there is no call to specify a value for the "SSL configuration name" key:

```
java.util.Properties prop = new java.util.Properties();
prop.put("connection security level", "10");
prop.put("key recovery password",keyPassword);
```

When working with an IRISDataSource object, if you want to specify an unnamed configuration, simply do not call **setSSLConfigurationName**.

6

Configuring .NET Clients to Use TLS with InterSystems IRIS

InterSystems IRIS® data platform supports TLS connections from .NET clients.

To establish a .NET connection that uses TLS:

1. If you have not done so already, configure the InterSystems IRIS superserver to use TLS so it can accept TLS connections from the .NET client.
2. Create a TLS configuration for the .NET client.
3. Ensure that you have installed any relevant CA certificates for verifying the server certificate. The location for these is the current user's certificate store (Certificates – Current User\Trusted Root Certification Authorities).
4. Establish a connection to a server, based on the format of the connection string as described in the Creating a Connection section of "Connecting to the InterSystems Database". In addition to the name-value pairs for the server, port, and namespace, include the *SSL* keyword and specify its value as `true`. For example, a connection that uses TLS protection might have a connection string of the form:

    ```
    IrisConnect.ConnectionString =
        "Server=localhost; Port=1972; Namespace=TESTNAMESPACE; SSL=true;"
        + "Password=SYS; User ID=_SYSTEM;";
    ```

    The `true` value of the SSL keyword specifies that TLS secures the client-server connection (by authenticating the InterSystems IRIS server to the .NET client and, optionally, authenticating the client to the server). Once the secure connection is established, the InterSystems IRIS server uses the User ID and Password keywords to authenticate the identity of the user connecting through the .NET client. (Note that the connection string does not specify anything related to mutual authentication; it merely specifies a server, which in turn may request or require client authentication.)

7

Configuring Studio to Use TLS with InterSystems IRIS

You can configure Studio to use a TLS connection. The process involves several steps:

1. On the instance acting as a TLS server and accepting the connection from Studio:
    a. Set up the `%SuperServer` TLS configuration. For more information on this process, see "Configuring the InterSystems IRIS Superserver to Use TLS."
    b. Enable TLS connections for superserver clients (because Studio is a superserver client).

       Specifically, on the **System-wide Security Parameters** page (**System Administration** > **Security** > **System Security** > **System-wide Security Parameters**), for the **Superserver SSL/TLS Support** field, choose **Enabled**.
2. On the Windows machine where Studio is running (which is acting as a TLS client), configure the settings file for the connection from Studio to the TLS server instance. For more information on this process, see Connecting from a Windows Client Using a Settings File.

8

Connecting from a Windows Client Using a Settings File

If you are on Windows and are using Studio, ODBC, or the Terminal as a TLS client, you can use a settings file to configure connections and configurations. This mechanism is available even if there is no instance of InterSystems IRIS® data platform on the host.

8.1 Overview of the Process

To use a settings file:

1. Get the certificate authority (CA) certificate for the server. Store it on disk and note the location — you will use it later.
2. Create a file containing connection definitions and configuration definitions, as described in the About the Settings File section.
3. Name the file SSLDefs.ini and place it in the InterSystems\IRIS directory in the directory for 32-bit common program files. Typically, this is the C:\Program Files (x86)\Common Files\InterSystems\IRIS\ directory; if you need to locate the directory, check the value of the Windows environment variable *CommonProgramFiles(x86)* on 64-bit Windows or *CommonProgramFiles* on 32-bit Windows.

By creating the file and placing it in this location, it will automatically be used when you connect to a host and a port that match one of the connections listed in the file.

Note: Use of the settings file (SSLDefs.ini) has the following restrictions:

1. The settings file is only for connections that use the irisconnect.dll or irisconnect64.dll executable (which are for 32-bit and 64-bit machines, respectively). Connections that use other mechanisms (such as for ADO) do not use the settings file.
2. Connections from a Windows client to InterSystems IRIS that use the settings file do not support Kerberos authentication.

8.2 About the Settings File

A settings file holds specifications for both connections to TLS servers and the TLS configurations that those connections use. For each Windows host that is a TLS client, a single file holds all its connections and configurations. The necessary information to create a file is:

- The Syntax of the Settings File
- Connection Properties
- Configuration Properties

8.2.1 The Syntax of the Settings File

The settings file contains one ore more *connection definitions* and one or more *configuration definitions*:

- Each definition begins with an identifier for the connection or configuration. This appears in brackets on its own line, such as:

  ```
  [MyConfiguration]
  ```

  The identifier can include spaces and punctuation, such as:

  ```
  [MyOtherConfiguration, which connects outside of my local network]
  ```

- Each definition ends either with the next bracketed identifier or the end of the file.
- Each definition includes multiple key-value pairs. All of these use the syntax:

  ```
  key=value
  ```

- The group of key-value pairs specify the properties of a connection definition or configuration definition.
- The value in each key-value pair appears unquoted.

8.2.2 Connection Definitions

Each settings file contains one or more *connection definitions*, each of which specifies the properties a TLS connection and matches that connection to a TLS configuration. The first line of a connection definition is its identifier, which appears in brackets. After the identifier, there are multiple lines specifying information about the TLS server and the connection to it:

Address

Required. The address of the TLS server. This can be an IP address, an unqualified host name in the local domain, or a fully-qualified hostname. (Note: The client only uses the specified configuration if the values of both Address and either Port or TelnetPort match the server to which the client application is connecting.)

Port

Required. The port number on which the TLS server accepts connections. (Note: The client only uses the specified configuration if the values of both Address and either Port or TelnetPort match the server to which the client application is connecting.)

TelnetPort

The port number on the TLS server that accepts TLS-protected connections for InterSystems Telnet. If you do not specify this value, connections using InterSystems Telnet do not support TLS. (Note: The client only uses the specified configuration if the values of both Address and either Port or TelnetPort match the server to which the client application is connecting.)

SSLConfig

Required. The TLS configuration that the client uses when connecting to the server specified in this definition. Each configuration is defined in its own section.

8.2.3 Configuration Definitions

Each settings file contains one or more *configuration definitions*, each of which specifies the properties of a TLS configuration; for more information on TLS configurations, see "About Configurations." The first line of a configuration definition is its identifier, which appears in brackets; if the configuration identifier appears as the value of a connection definition's SSLConfig property, the connection uses the configuration to specify its behavior. After the identifier, there are multiple lines specifying the value of each of the configuration's properties:

`Protocols`

Deprecated. Use `TLSMinVersion` and `TLSMaxVersion` instead.

The versions of the TLS protocol(s) that the configuration supports, where each version of the protocol has a numeric value as listed in `TLSMinVersion` and `TLSMaxVersion`. To specify support for multiple versions of the protocol, use the sum of their values. Hence, to specify support for TLS v1.1 and TLS v1.2, use a value of 24.

This property is equivalent to the **Protocols** field in the TLS configuration page in the Management Portal.

`TLSMinVersion`

Required for configurations that support v1.3. The earliest version of the TLS protocol that this configuration supports, where each version of the protocol has a numeric value and supported versions are:

- TLS v1 — 4
- TLS v1.1 — 8
- TLS v1.2 — 16
- TLS v1.3 — 32

This property is equivalent to the **Minimum Protocol Version** field in the TLS configuration page in the Management Portal.

`TLSMaxVersion`

Required for configurations that support v1.3. The most recent version of the TLS protocol that this configuration supports, where each version of the protocol has a numeric value and supported versions are:

- TLS v1 — 4
- TLS v1.1 — 8
- TLS v1.2 — 16
- TLS v1.3 — 32

This property is equivalent to the **Maximum Protocol Version** field in the TLS configuration page in the Management Portal.

`VerifyPeer`

Required. Whether or not the client requires the verification of the certificate of the server to which it is connecting:

- 0 — Does not require (and does not perform) peer verification; the connection is established under all circumstances.
- 1 — Requires peer verification; the connection is established only if verification succeeds. This is the recommended value; if you choose this value, you must specify a value for the CAFile property.

This property is equivalent to the **Server certificate verification** field in the TLS configuration page in the Management Portal.

`VerifyHost`

Whether or not the client checks if the Common Name or subjectAlternativeName fields of the server's certificate match the host name or IP address as specified in the connection definition:

- 0 — Does not check.
- 1 — Checks.

This property does not have an equivalent in the Management Portal. However, it is the same type of check as the SSLCheckServerIdentity property of the %Net.HttpRequest class.

`CipherList`

Required for configurations that support connections using TLS v1.2 or earlier. The set of cipher suites that the client supports for encryption and hashing. For information on this property's syntax, see the OpenSSL documentation on the ciphers command.

The default value is `ALL:!aNULL:!eNULL:!EXP:!SSLv2`, and InterSystems strongly suggests using this value. For more information about this syntax in InterSystems IRIS, see Enabled Cipher Suites Syntax.

This property is equivalent to the **Enabled ciphersuites** field in the TLS configuration page in the Management Portal.

`Ciphersuites`

Required for configurations that support connections using TLS v1.3. The set of ciphers that the client supports for encryption and hashing. For information on this property's syntax, see the OpenSSL documentation on the ciphers command.

InterSystems strongly recommends using a value of `TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256`, which is the default. For more information about this syntax in InterSystems IRIS, see Enabled Cipher Suites Syntax.

`CertFile`

The absolute path and name of the file that contains the client's trusted certificate authority (CA) file; if the client does not have a CA, do not specify a value for this property. If specified, this is an X.509 certificate(s) in PEM format and can include a certificate chain. For information on how this value is used, see Establishing the Required Certificate Chain. (Note that certificates from the Windows Certificate Export Wizard must be in PEM-encoded X.509 format, not the default of DER encoded binary X.509.)

This property is equivalent to the **File containing this client's certificate** field in the TLS configuration page in the Management Portal.

`KeyFile`

The absolute path and name of the configuration's private key file; if the client does not have a private key, do not specify a value for this property.

This property is equivalent to the **File containing associated private key** field in the TLS configuration page in the Management Portal.

`Password`

The password for decrypting the configuration's private key. If you are using a private key with a password, this property is required; if you are not using a certificate for the client or if the private key does not have a password, do not specify a value for this property. (If the private key is password-protected and you do not provide a value here, InterSystems IRIS cannot decrypt and use the private key.)

This property is equivalent to the **Private key password** field in the TLS configuration page in the Management Portal.

`KeyType`

If the configuration has a private key and certificate, the format in which the configuration's private key is stored:

- DSA — 1
- RSA — 2

This property is equivalent to the **Private key type** field in the TLS configuration page in the Management Portal.

`CAfile`

Required. The absolute path and name of the file that contains the server's trusted certificate authority (CA) file. This is an X.509 certificate(s) in PEM format. Note that:

- If you have specified a VerifyPeer value of 1, you must provide this value.
- This is the certificate for CA of the server to which you are connecting, *not* the certificate for your CA.

This property is equivalent to the **File containing trusted Certificate Authority certificate(s)** field in the TLS configuration page in the Management Portal. However, unlike the Portal, it does not support the use of the `%OSCertificateStore` string.

8.3 A Sample Settings File

The following sample file defines two connections and two configurations:

```
[MyServer1 TLS to an InterSystems IRIS instance with TLS-protected InterSystems Telnet]
Address=myserver1
Port=57777
TelnetPort=23
SSLConfig=TLSConfig

[MyServer2 TLS to an InterSystems IRIS instance using TLSv1.2]
Address=myserver2.myexample.com
Port=57777
SSLConfig=TLSv1.2only

[TLSConfig]
TLSMinVersion=16
TLSMaxVersion=32
CipherList=ALL:!aNULL:!eNULL:!EXP:!SSLv2
Ciphersuites=TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256
KeyType=2
```

```
VerifyPeer=1
Password=
CertFile=c:\InterSystems\certificates\nopwclicert.pem
KeyFile=c:\InterSystems\certificates\nopwclikey.pem
CAfile=c:\InterSystems\certificates\cacert.pem

[TLSv1.2only]
TLSMinVersion=16
TLSMaxVersion=16
CipherList=ALL:!aNULL:!eNULL:!EXP:!SSLv2
KeyType=2
VerifyPeer=1
Password=
CertFile=c:\InterSystems\certificates\nopwclicert.pem
KeyFile=c:\InterSystems\certificates\nopwclikey.pem
CAfile=c:\InterSystems\certificates\cacert.pem
```

8.4 How It Works

Important: This section describes how InterSystems products use a settings file to establish a TLS connection. By describing the mechanisms in use, it includes alternate means of creating a TLS connection. InterSystems recommends that you use the standard approach described above, rather than the alternatives mentioned here.

InterSystems IRIS uses the settings file as follows:

1. When you attempt to establish a TLS connection, the InterSystems IRIS TCP/IP client connection library locates the settings file containing connection definitions and configurations. This file is irisconnect.dll on 32-bit machines and irisconnect64.dll on 64-bit machines. To do this:
   a. It checks the Windows registry for any TLS connection definitions.
   b. If there are no connection definitions in the registry, the library attempts to locate any TLS configurations that are stored in a settings file.
   c. If the *ISC_SSLconfigurations* environment variable exists, the library uses the value of that variable as the full path and file name of the settings file.

      Note: If you need to define the value of the *ISC_SSLconfigurations* environment variable, you may need administrator permissions.

   d. If the *ISC_SSLconfigurations* environment variable does not exist, the library uses the SSLdefs.ini file in the InterSystems\IRIS directory under the 32-bit common program files directory identified by the Windows environment variables *CommonProgramFiles(x86)* on 64-bit Windows or *CommonProgramFiles* on 32-bit Windows.
2. Once it has located the settings file, the library locates the relevant connection definition for the connection you are attempting to establish.

   To do this, it searches the sections of the file for one that contains Address and Port properties that match those of the connection you are attempting to establish. When it locates such a section, it uses the value of the *SSLConfig* property there to locate the matching TLS configuration section.
3. In the specified TLS configuration section, the library uses the values of the configuration properties to specify the type of connection to initiate with the server.

9

Configuring InterSystems IRIS to Use TLS with Mirroring

For general information about InterSystems IRIS® data platform support for mirroring, see Mirroring.

9.1 About Mirroring and TLS

To provide security within a mirror, you can configure its nodes to use TLS. This provides for both authentication of one node to another, and for encrypted communication between nodes. As sensitive data passes between the failover members (and to an async member), it is recommended to encrypt the communication to prevent data theft or alteration over the network. Additionally, since a failover member has the ability to request an ISCAgent to take action on another InterSystems IRIS system (such as to request journal file information or force InterSystems IRIS down), it is important to protect such communication between the failover members of a mirror (and their corresponding ISCAgent processes).

Note: If the failover members use database (or journal) encryption, then TLS is required for communications between them and with any async members. (Specifically, InterSystems IRIS checks if either member has an encryption key activated; if so, the instance requires that the user enable TLS with mirroring.) For more details on database encryption and journal file encryption, see Encryption Guide.

To both participate in mirroring (either as a failover member or as an async member) and use TLS, an instance must have two InterSystems IRIS TLS configurations – one of type server and the other of type client; each of these must have an X.509 TLS certificate issued by a trusted Certificate Authority. InterSystems recommends that each mirror host has its own set of certificates and that the certificates contain a unique identifier in the Common Name (CN) component of the certificate, such as the fully qualified domain name (FQDN) of the instance plus the member's InterSystems IRIS node name; because the CN is a field in a certificate's distinguished name (DN), establishing a unique CN ensures that the certificate's DN uniquely identifies the member. To create an instance's mirroring configurations, follow the procedure in the next section.

When TLS is enabled, the following actions occur:

1. Server authentication: When the client connects to the server, it requires the server to authenticate itself. This authentication verifies that the DN for the server's certificate matches the DN for a system configured in the client's mirror configuration. If there is no match, the client drops the connection.
2. Client authentication: When the server accepts a connection from a client, it requires the client to authenticate itself. This authentication also verifies that the DN for the client's matches the DN for a system configured in the server's mirror configuration. Again, if there is no match, the server drops the connection.

3. Encryption: The TLS protocol automatically uses the server's certificate to establish an encrypted channel between the client and the server, so that any data passing through this channel is encrypted and thereby secured.

InterSystems strongly recommends using TLS with a mirror.

Note on Configuring an Async Member with TLS

If a mirror uses TLS, then in addition to enabling TLS for the mirror and creating the configurations for each member (described in the following section), there are special steps that must be taken when configuring the second failover member or an async member; for more information, see Authorize the Second Failover Member or Async (TLS Only). Specifically, for each failover member, on the **Mirror Monitor** page, you need to enter the DN (distinguished name) in the **ID listed as DN in member's X.509 credentials** field; you can copy the value of the DN from **X.509 Distinguished Name** field of the **Join as Async** page (**System Administration** > **Configuration** > **Mirror Settings** > **Join as Async**) for the async member. (InterSystems IRIS populates the **X.509 Distinguished Name** field based on the information in the async member's certificate, making this field unavailable for manual editing.)

Note on Disabling TLS for a Mirror

To disable TLS for an existing mirror, disable it on the primary member.

Important: Use of TLS with mirroring is highly recommended. Disabling TLS for a mirror is strongly discouraged.

9.2 Create and Edit a TLS Configuration for a Mirror

To use TLS with a mirror, each member (failover or async) uses a pair of TLS configurations that are called `%MirrorClient` and `%MirrorServer`; the Portal allows you to create and edit these configurations.

Note: These configurations must already exist on each member when TLS is enabled for the mirror.

9.2.1 Create a TLS Configuration for a Mirror Member

To create the configurations, the procedure is:

1. Enable mirroring for that instance of InterSystems IRIS if it is not already enabled. To do this, use the **Edit Service** page for the **`%Service_Mirror`** service; on this page, select the **Service Enabled** check box. You can reach this page by either of two paths:
   - On the **Mirror Settings** page (**System Administration** > **Configuration** > **Mirror Settings**), select **Enable Mirror Service**.
   - On the **Services** page (**System Administration** > **Security** > **Services**), select **%Service_Mirror**.
2. Go to the **Create SSL/TLS Configurations for Mirror** page. You can do this either by:
   - On the **SSL/TLS Configurations** page (**System Administration** > **Security** > **SSL/TLS Configurations**) select **Create Configurations for Mirror**.
   - On the **Create Mirror** page (**System Administration** > **Configuration** > **Mirror Settings** > **Create Mirror**) select **Set up SSL/TLS**.
3. On the **Create SSL/TLS Configurations for Mirror** page, complete the fields on the form. The fields on this page are a subset of those on the **New SSL/TLS Configuration** page (as described in Create or Edit a TLS Configuration). Since this page creates both server and client configurations that mirroring automatically enables (%MirrorClient and %MirrorServer), there are no **Configuration Name**, **Description**, or **Enabled** fields; also, for the private-key password, this page allows you to enter or replace one (**Enter new password**), specify that none is to be used (**Clear password**), or leave an existing one as it is (**Leave as is**).

Since both configurations need the same X.509 credentials, completing this form saves both configurations simultaneously. The fields described in Create or Edit a TLS Configuration on this page are:

- File containing trusted Certificate Authority certificate(s)

  Note: This file must include the certificate(s) that can be used to verify the X.509 certificates belonging to other mirror members. If the file includes multiple certificates, they must be in the correct order, as described in Establishing the Required Certificate Chain, with the current instance’s certificate first.

- This server's credentials:
  - **File containing associated private key**
  - **Private key type**
  - **Private key password**
  - **Private key password (confirm)**
- Cryptographic settings:
  - **Minimum Protocol Version**
  - **Maximum Protocol Version**

    Enabled cipherlist (TLSv1.2 and below)
  - **Enabled ciphersuites (TLSv1.3)**
  - **Diffie Hellman Bits**
- OCSP Settings:
  - **OCSP Stapling**

Once you complete the form, click **Save**.

For general information about configuring mirror members, see Creating a Mirror.

9.2.2 Edit TLS Configurations for a Mirror Member

If you have already created a member’s %MirrorClient and %MirrorServer configurations, you can edit them on the **Edit SSL/TLS Configurations for Mirror** page (**System Administration** > **Security** > **SSL/TLS Configurations**; click **Edit Configurations for Mirror**). This page displays the same fields as the **Create SSL/TLS Configurations for Mirror** page, as described in the previous section.

9.2.3 Special Considerations for Certificates for Mirror Members

When using TLS with mirroring, the %MirrorClient and %MirrorServer configurations must use the same certificate and private key. Hence, the certificate in use by both configurations must be usable as both a server and a client certificate.

There are certain certificate extensions that are specific to TLS clients or servers. Because the certificate in use with mirroring must be able to serve both uses (as both a client and a server), if any of these extensions appear in a certificate, then the extensions for client and server must both be present. For example, this is true for the Key Usage and Extended Key Usage extensions. If the Key Usage extension is present, then it must specify both of the following:

- The Digital Signature key usage (for clients)
- The Key Encipherment key usage (for servers)

Similarly, if the Extended Key Usage extension is present, then it must specify both:

- The Client Authentication key usage
- The Server Authentication key usage

If both extensions are present, then each must specify both values. Of course, it is also valid to have neither extension present.

If a certificate only specifies one value (either client or server), the TLS connection for mirroring fails with an error such as:

```
error:14094413:SSL routines:SSL3_READ_BYTES:sslv3 alert unsupported certificate
```

The way to eliminate this error depends on how you obtained your certificates:

- If you are using self-signed certificates, create new certificates (such as with the OpenSSL library) that adhere to these conditions.
- If you are using a commercial certificate authority tool (such as Microsoft Certificate Services), create new certificates that adhere to these conditions and use the tool to sign your certificate signing requests (CSRs).
- If you are purchasing certificates from a commercial certificate authority (such as VeriSign), include a request along with your CSRs that they adhere to these conditions.

10

Configuring InterSystems IRIS to Use TLS with TCP Devices

This section describes how to use TLS with an InterSystems IRIS® data platform TCP connection. The process is:

1. Creating a TLS configuration that specifies the characteristics you want.
2. Opening a TCP connection or open a socket for accepting such connections.
3. Securing the connection using TLS. This can occur either as part of opening the connection/socket or afterwards.

How you invoke the InterSystems IRIS TLS functionality depends on whether you are using InterSystems IRIS as a client or server and whether you are creating an initially-secured TCP connection or adding TLS to an existing connection.

This section addresses the following topics:

- Configure a Client to Use TLS with a TCP Connection
- Configure a Server to Use TLS with a TCP Socket

10.1 Configure a Client to Use TLS with a TCP Connection

To establish a secure connection from a client, the choices are:

- Open a TLS-secured TCP Connection from a Client
- Add TLS to an Existing TCP Connection

10.1.1 Open a TLS-secured TCP Connection from a Client

In this scenario, InterSystems IRIS is part of the client and the TCP connection uses TLS from its inception. The procedure is:

1. Make sure that the configuration you wish to use is available. If it was created before InterSystems IRIS was last started, it is activated and ready for use; otherwise, you can create a new one or edit an existing one.
2. Open a TCP Connection Using TLS.

If InterSystems IRIS is a client, then it connects to the server via the client application. The connection uses the specified configuration to determine its TLS-related behavior.

10.1.1.1 Open a TCP Connection Using TLS

This involves opening a named connection that uses TLS and communicates with a particular machine and port number. The procedure is:

1. Specify the device that you are connecting to:

   ObjectScript

   ```
   Set MyConn = "|TCP|1000"
   ```

   The TCP string specifies that this is a TCP device. For more information on initiating a TCP connection, see OPEN Command for TCP Devices.

2. Open the connection, specifying the use of TLS with the `/TLS` parameter.

   ObjectScript

   ```
   OPEN MyConn:(SvrID:1000:/TLS="MyCfg")
   ```

   where

   - *MyConn* is the device previously specified
   - *SvrID* can be a string that is a resolvable DNS name or an IP address
   - *MyCfg* is a saved (and activated) TLS configuration

   This call opens a TCP connection to the loopback processor (that is, the local machine) on port 1000 using TLS. It uses TLS according to the characteristics specified by the MyCfg configuration.

   Optionally, the call can include a password for the private key file:

   ```
   OPEN MyConn:(SvrID:1000:/TLS="MyCfg|MyPrivateKeyFilePassword")
   ```

   Here, all the arguments are as above and *MyPrivateKeyFilePassword* is the actual password.

   Important: The ability to include a password when Open a TCP Connection Using TLS is for real-time interactive use only. You should *never* store a private key password persistently without protecting it. If you need to store such a password, use the PrivateKeyPassword property of the Security.SSLConfigs class.

   For more information on opening a TCP device, see OPEN and USE Command Keywords for TCP Devices.

Once the connection is established, you can then use it in the same manner as any other TCP connection.

10.1.2 Add TLS to an Existing TCP Connection

This scenario assumes that the TCP connection has already been established. The procedure is:

1. Make sure that the configuration you wish to use is available. If it was created before InterSystems IRIS was last started, it is activated and ready for use; otherwise, you can create a new one or edit an existing one.
2. Securing the existing TCP connection using TLS.

10.1.2.1 Secure an Existing TCP Connection Using TLS

This involves adding TLS to an already-existing connection to a particular machine and port number. The procedure is:

1. Determine the name of the device to which there is a connection. For example, this might have been established using the following code:

   ```
   SET MyConn="|TCP|1000"
   OPEN MyConn:("localhost":1000)
   ```

   The TCP string specifies that this is a TCP device. For more information on initiating a TCP connection, see OPEN Command for TCP Devices.

2. Specify the use of TLS as follows with the `/TLS` parameter:

   ```
   USE MyConn:(::/TLS="MyCfg")
   ```

   where

   - *MyConn* is the device previously specified
   - *MyCfg* is a TLS configuration

   Optionally, the call can include a password for the private key file:

   ```
   USE MyConn:(::/TLS="MyCfg|MyPrivateKeyFilePassword")
   ```

   Here, all the arguments are as above and *MyPrivateKeyFilePassword* is the actual password.

   Important: The ability to include a password when securing an existing TCP connection using TLS is for real-time interactive use only. You should *never* store a private key password persistently without protecting it. If you need to store such a password, use the PrivateKeyPassword property of the Security.SSLConfigs class.

   For more information on opening a TCP device, see OPEN and USE Command Keywords for TCP Devices.

Having added TLS security to the connection, you can continue to use it in the same manner as before.

10.2 Configure a Server to Use TLS with a TCP Socket

To enable a socket to require a secure connection from a client, you can either:

- Open a TCP socket specifying that this connection requires TLS.
- Establish the requirement for the use of TLS on an already-existing socket.

10.2.1 Establish a TLS-secured Socket

In this scenario, InterSystems IRIS is the server and the TCP socket uses TLS from its inception. The procedure is:

1. Make sure that the configuration you wish to use is available. If it was created before InterSystems IRIS was last started, it is activated and ready for use; otherwise, you can create a new one or edit an existing one.
2. Open a TCP socket that requires the use of TLS.

This socket requires the use of TLS from clients connecting to it. When a client attempts to connect to the server, the server attempts to negotiate a connection that uses TLS. If this succeeds, the connection is available for normal use and communications are secured using the negotiated algorithm. If it fails, there is no connection available for the client.

10.2.1.1 Open a TCP Socket Requiring TLS

To open a socket that requires TLS, the procedure is:

1. Specify the device that is accepting connections:

   ObjectScript

   ```
   SET MySocket = "|TCP|1000"
   ```

   The TCP string specifies that this is a TCP device. For more information on initiating a TCP connection, see OPEN Command for TCP Devices.

2. Open the connection, specifying the use of TLS with the /TLS parameter.

   ObjectScript

   ```
   OPEN MySocket:(:1000:/TLS="MyCfg")
   ```

   Optionally, the call can include a password for the private key file:

   ```
   OPEN MySocket:(:1000:/TLS="MyCfg|MyPrivateKeyFilePassword")
   ```

   This call opens a TCP socket on port 1000 using TLS. For more information on opening a TCP device, see OPEN and USE Command Keywords for TCP Devices.

   Important: The ability to include a password when opening a TCP connection using TLS is for real-time interactive use only. You should *never* store a private key password persistently without protecting it. If you need to store such a password, use the PrivateKeyPassword property of the Security.SSLConfigs class.

10.2.2 Add TLS to an Existing Socket

This scenario assumes that a connection to the TCP socket has already been established. The procedure is:

1. Make sure that the configuration you wish to use is available. If it was created before InterSystems IRIS was last started, it is activated and ready for use; otherwise, you can create a new one or edit an existing one.
2. Use TLS to secure the existing TCP connection to the socket.

10.2.2.1 Secure an Existing TCP Connection to the Socket Using TLS

This involves adding TLS to an already-existing connection to a socket on a particular machine and port number. The procedure is:

1. Determine the name of the device on which the socket is open. For example, this might have been established using the following code:

   ```
   SET MySocket = "|TCP|1000"
   OPEN MySocket:(:1000)
   ```

   The TCP string specifies that this is a TCP device. For more information on initiating a TCP connection, see OPEN Command for TCP Devices.

2. Specify the use of TLS as follows with the `/TLS` parameter:

   ```
   USE MySocket:(::/TLS="MyCfg")
   ```

   where

- *MySocket* is the device previously specified
- *MyCfg* is a TLS configuration

Optionally, the call can include a password for the private key file:

```
USE MySocket:(::/TLS="MyCfg|MyPrivateKeyFilePassword")
```

For more information on opening a TCP device, see OPEN and USE Command Keywords for TCP Devices.

Important: The ability to include a password when securing an existing TCP connection using TLS is for real-time interactive use only. You should *never* store a private key password persistently without protecting it. If you need to store such a password, use the PrivateKeyPassword property of the Security.SSLConfigs class.

Having added TLS security to the socket, you can continue the connection to it in the same manner as before.

11

Configuring the Web Gateway to Connect to InterSystems IRIS Using TLS

You can use TLS to set up a secure, encrypted channel between the Web Gateway and the InterSystems IRIS® data platform server. To do this, you need a TLS certificate and private key that represents the Gateway. The Gateway can then establish an encrypted connection to the InterSystems IRIS server (which has its own certificate and private key), so that all information is transmitted through the connection.

Note: For information on setting up a connection between the Web Gateway and the InterSystems IRIS server that is protected by Kerberos, see Setting Up a Kerberized Connection from the Web Gateway to InterSystems IRIS.

The procedure is:

1. If there is not already a `%SuperServer` TLS configuration associated with the InterSystems IRIS server, create one as described in Create or Edit a TLS Configuration.
2. On the Portal’s **System-wide Security Parameters** page (**System Administration > Security > System Security> System-wide Security Parameters**), for the **Superserver SSL/TLS Support** choice, select **Enabled** or **Required**. For more details on these settings, see System-wide Security Parameters.
3. Go to the Web Gateway’s **Server Access** page (**System Administration > Configuration > Web Gateway Management**).
4. On that page, under **Configuration**, select **Server Access**.
5. Next, select **Edit Server** and click **Submit**. This displays the configuration page for the Web Gateway.
6. On this page, configure the Web Gateway to use TLS. Specifically, for the **Connection Security Level** field, select **SSL/TLS**. You must specify values for the **SSL/TLS Protocol** and **SSL/TLS CA Certificate File** fields. The other fields may be required or optional depending on other settings. The **SSL/TLS Certificate File** and **SSL/TLS Private Key File** are required if **Require peer certificate verification** is selected. If including a SSL/TLS private key file, you must also specify a value for the **SSL/TLS Key Type**. Additionally, if the certificate or private key file require a password, then the password must be supplied in **SSL/TLS Private Key Password**.

   For more details on the fields on this page, see the Configuring Server Access section of “Web Gateway Operation and Configuration”.

12

Establishing the Required Certificate Chain

For a connection to be successfully established using a cipher suite that uses certificates and keys, the client must be able to verify the server’s certificate chain from the server’s own certificate to a self-signed certificate from a trusted certificate authority (CA), including intermediate certificates (if any). If the server is authenticating the client user, then the server must also be able to verify the client user’s certificate chain from the client user’s own certificate to a trusted CA’s self-signed certificate, including intermediate certificates (if any).

Since authentication can be bidirectional, the requirements for certificate chains refer to the verifying entity (the side requiring the authentication) and the verified entity (the side being authenticated), rather than the client and the server.

For authentication to be possible, the following conditions must be met:

- The verifying entity must have access to all the certificates that constitute the certificate chain from the verified entity’s own certificate to a trusted CA’s self-signed root certificate. The certificates in the chain are obtained from the combination of the verified entity’s certificate file (the certificates are sent as part of the handshake protocol) and the verifying entity’s trusted CA certificate file.
- The verifying entity must have the trusted CA’s self-signed root certificate in its CA certificate file.
- The verified entity’s own certificate must be the first entry in its certificate file.
- All intermediate CA certificates must be present.
- The certificates in the certificate chain may be divided between the verified entity’s certificate file and the verifying entity’s trusted CA certificate file. However, each part must be a contiguous partial certificate chain, as described in the following example.

Suppose there are:

- A verified entity (named “VE”) with a certificate signed by the certificate authority named “ICA1.”
- A certificate for “ICA1” signed by the certificate authority “ICA2,” and a certificate for “ICA2” signed by “RootCA”.
- A trusted CA (named “RootCA”) with a self-signed root certificate.

The following are valid distributions of certificates between the verified entity and the verifying entity:

Table 12–1: Valid Certificate Distribution Schemes

Certificates in the Verified Entity's Certificate File	Certificates in the Verifying Entity's Trusted CA Certificate File
VE	ICA1, ICA2, RootCA
VE, ICA1	ICA2, RootCA
VE, ICA1, ICA2	RootCA

Note that it is *not* valid to have VE and ICA2 in the verified entity's certificate file and ICA1 and RootCert in the verifying entity's trusted CA certificate file