CAPLIN

# KeyMaster 5.0

## Overview

December 2009

CONFIDENTIAL

# Contents

# 1 Preface

## 1.1 What this document contains

This document provides an overview of the Caplin KeyMaster product, version 5.0.

It explains:

- What KeyMaster is and what it can be used for.
- The architecture of the product.
- How the product fits into the overall Caplin product architecture and third party/customer systems.
- Key concepts relating to the product.

It also gives some examples of how the product can be used in real business situations.

### About Caplin document formats

This document is supplied in three formats:

- Portable document format (*.PDF* file), which you can read on-line using a suitable PDF reader such as Adobe Reader®. This version of the document is formatted as a printable manual; you can print it from the PDF reader.
- Web pages (*.HTML* files), which you can read on-line using a web browser. To read the web version of the document navigate to the *HTMLDoc_m_n* folder and open the file *index.html*.
- Microsoft HTML Help (*.CHM* file), which is an HTML format contained in a single file.
  To read a *.CHM* file just open it – no web browser is needed.

**For the best reading experience**

On the machine where your browser or PDF reader runs, install the following Microsoft Windows® fonts: Arial, Courier New, Times New Roman, Tahoma. You must have a suitable Microsoft license to use these fonts.

**Restrictions on viewing .CHM files**

You can only read *.CHM* files from Microsoft Windows.

Microsoft Windows security restrictions may prevent you from viewing the content of *.CHM* files that are located on network drives. To fix this either copy the file to a local hard drive on your PC (for example the Desktop), or ask your System Administrator to grant access to the file across the network. For more information see the Microsoft knowledge base article at
http://support.microsoft.com/kb/896054/.

## 1.2 Who should read this document

This document is intended for anyone who requires an introduction to KeyMaster.

Typical readers include:

- Technical Managers
- System Architects
- System Administrators
- Operators
- Software Developers

## 1.3 Related documents

- **KeyMaster Administration Guide**

  Describes how to install, configure and run the standard version of KeyMaster.

- **KeyMaster Java API Reference**

  Defines KeyMaster Java™ classes and interfaces that can be called and extended to produce customized versions of the KeyMaster Java servlet.

- **KeyMaster.NET API Reference**

  Defines KeyMaster.NET classes and interfaces that can be used to implement a Microsoft .NET application that generates KeyMaster user credentials tokens.

- **Liberator Administration Guide**

  Describes how to install and configure the Caplin Liberator server. It includes full reference information for the Liberator configuration.

- **Auth SDK Overview**

  Gives an overview of the Authentication Modules ("Auth Modules") that come with Caplin Liberator. Developers can use the Auth Module SDK to create customized modules for controlling user authentication and object access permissions.

- **XML Auth Administration Guide**

  Describes the XML Auth Module. XML Auth enables programmers and system administrators to use XML to create their own object access permission structures, and control entitlement to objects held on Caplin Liberator.

- **Liberator Authentication C API Reference**

  Describes how to implement custom Authentication Modules for Liberator, in C code.

- **JavaAuth API Reference**

  Describes the library of classes (javaauth) that enables developers to create custom Authentication Modules for Liberator in Java.

- **StreamLink for Browsers API Reference**

  The API reference documentation for StreamLink for Browsers. In release 4.5.2 and upwards, the section on "Using SL4B With KeyMaster" explains how StreamLink for Browsers can be configured to use KeyMaster for logging in to the Liberator.

- **StreamLink for Java API Reference**

  The API reference documentation for StreamLink for Java. The section on "Caplin KeyMaster Integration" explains how KeyMaster authentication can be integrated into web client applications that use StreamLink for Java.

- **StreamLink.NET API Reference**

  The API reference documentation for StreamLink .NET.

- **StreamLink for Silverlight API Reference**

  The API reference documentation for StreamLink for Silverlight.

## 1.4 Typographical conventions

The following typographical conventions are used to identify particular elements within the text.

| *Type* | *Uses* |
|---|---|
| **aMethod** | Function or method name |
| *aParameter* | Parameter or variable name |
| */AFolder/Afile.txt* | File names, folders and directories |
| `Some code;` | Program output and code examples |
| The `value=10` attribute is... | Code fragment in line with normal text |
| Some text in a dialog box | Dialog box output |
| `Something typed in` | User input – things you type at the computer keyboard |
| **XYZ Product Overview** | Document name |
| ◆ | Information bullet point |
| ■ | Action bullet point – an action you should perform |

**Note:** Important Notes are enclosed within a box like this.
Please pay particular attention to these points to ensure proper configuration and operation of the solution.

**Tip:** Useful information is enclosed within a box like this.
Use these points to find out where to get more help on a topic.

## 1.5 Feedback

Customer feedback can only improve the quality of our product documentation, and we would welcome any comments, criticisms or suggestions you may have regarding this document.

Please email your feedback to documentation@caplin.com.

## 1.6 Acknowledgments



## 1.7 Open Source Software

This Caplin Xaqua component incorporates the following Open Source software:

| Open Source item | Use in KeyMaster | Further information |
|---|---|---|
| Public key encryption software from The Legion Of The Bouncy Castle. | Used in standard KeyMaster to generate encryption key pairs, and encrypt and decrypt digital signatures using these keys. | www.bouncycastle.org |
| OpenSSL | Cryptographic algorithms from the OpenSSL crypto library, and the OpenSSL commands for generating RSA key pairs and self-signed certificates. | www.openssl.org |

# 2 What is Caplin KeyMaster?

Caplin KeyMaster is software that integrates Caplin Liberator with an existing single sign-on 32 system, so that end users do not have to explicitly log in to the Liberator server in addition to logging in to the enterprise's single sign-on server.

KeyMaster implements a secure method of user authentication by means of a user credentials token that is digitally signed using public key encryption. It has two main tools that enable users to be authenticated: a key generator and a signature generator. These tools are described in the Architecture 8 section.

KeyMaster works in conjunction with other Caplin Xaqua components (see Other Caplin Xaqua components 20), and 3rd party software/customer developed software, including single sign-on and user permissions systems, web applications, and web application servers.

## 2.1 How KeyMaster works – a brief explanation

This explanation describes the steps involved in authenticating an end user through KeyMaster.
It assumes the user logs in to a Liberator server from a StreamLink enabled client.

1. The user logs in to an authenticating application server (not provided by Caplin):
   a) The server connects to a single sign-on system (not provided by Caplin) to check the validity of the login request.
   b) Typically, if the login is successful (the user is authenticated), the authenticating application server returns a cookie to the client, so that all subsequent interactions between it and the client do not require re-authentication.
2. When the user starts a StreamLink enabled application on the client, the StreamLink library automatically requests a user credentials token from the authenticating application server:
   a) The request is passed to the single-sign on system to check that it is for a valid and active user session.
   b) KeyMaster creates a user credentials token, typically containing a username, a timestamp, and a sequence number.
   c) KeyMaster uses the KeyMaster private key to digitally sign the user credentials token.
   d) The application server sends the signed user credentials token back to the client application (StreamLink).
3. StreamLink automatically logs the user into the Liberator:
   a) StreamLink sends a login request to the Liberator, passing it the user credentials token.
   b) The Liberator uses the KeyMaster public key to validate the signature of the token.
   c) The Liberator ensures the token has not timed out and is otherwise valid.
   d) The user is now logged in to the Liberator.

> **Tip:** For a more detailed explanation, see KeyMaster architecture 8 and How does KeyMaster work? 11

## 2.2 KeyMaster features

- Enables Liberator users to be authenticated via a single client application or web page sign-on.
- Uses a digitally signed user credentials token to provide secure and reliable user authentication.
- Utilizes public key cryptography.
- Standard Java implementation uses Bouncy Castle encryption.
- Removes the need to replicate across multiple servers the code that authenticates end users – only one end-user sign on point and associated user authentication system is required.
- Application web servers do not need to communicate with Liberator servers.
- Can be integrated with a secure key storage hardware module.
- Written in Java and Microsoft .NET.
- Can be deployed with web application servers that support Java servlets and Java version 1.4 or higher.
- Can be deployed in .NET environments, such as ASP.NET.
- The .NET-based and Java-based token generators work with any StreamLink-based client application (StreamLink.NET, StreamLink for Silverlight, StreamLink for Browsers, StreamLink for Java, and so on).
- Has been tested with the following web application servers:
  - Tomcat
  - JBoss
  - BEA Weblogic
  - IBM WebSphere
  - IIS for Windows Server hosting an ASP.NET KeyMaster deployment.
- Works in conjunction with the following Caplin Liberator Auth Modules:
  - XMLauth
  - cfgauth
  - Custom Auth Modules written using the javaauth or C Auth SDKs.
- Can be customized through configuration and Java or .NET coding – see the Customization 21 sections.

## 2.3 Concepts

To illustrate the concepts behind KeyMaster, consider a market data system where end users run web-based applications from PCs. These applications could be browser-based (running Java, Javascript, or Microsoft Silverlight™), or they could be native client applications. Some of the data required by these applications comes from one or more Caplin Liberator servers. Before the end users can run applications and access the data, they must be authenticated. For example, each user must log in by supplying valid user credentials, such as a user name and password.

The user's application will typically need to access various servers – web application servers, Liberator servers, and so on. These servers will also require that the user be authenticated before the application can grant access to them. As a result, the user is presented with multiple prompts for user credentials at different points in the dialogue with the system.

This is at best inconvenient for users, and can impose an administrative burden; for example, users have to manage multiple passwords which they therefore more easily forget, passwords will typically expire at different times, and there are likely to be multiple databases and locations where user credentials are held.

A solution to these issues is to implement a single sign-on system. Each end user is authenticated just once, for example, by entering a single user name and password, or by using some other more sophisticated form of authentication such as a smart card. The authentication applies to all the applications and data for which the user has access rights, so that during the rest of the user's session, access to the various servers that supply these applications and data is completely transparent to the user. No more log in prompts are issued for the remainder of the user's session; any further authentications required by the other servers are automatically performed in the background.

When integrated with an existing single-sign on facility, KeyMaster provides the infrastructure and mechanisms that allow end users to be automatically logged in to Liberator servers in a secure way. When a user's session needs to access a Liberator server for the first time, the client application or web page sends a request to the application server, which passes it on to KeyMaster.

Both KeyMaster and the single-sign on facility run under the control of the web application server, so KeyMaster can safely assume that the request, which has come via the web application server, is from an authenticated and authorized end user. KeyMaster responds to the request by generating a user credentials token that can be used to log in to the Liberator server. This token is sent back to the client application, which uses it instead of a normal password to log in to the Liberator server.

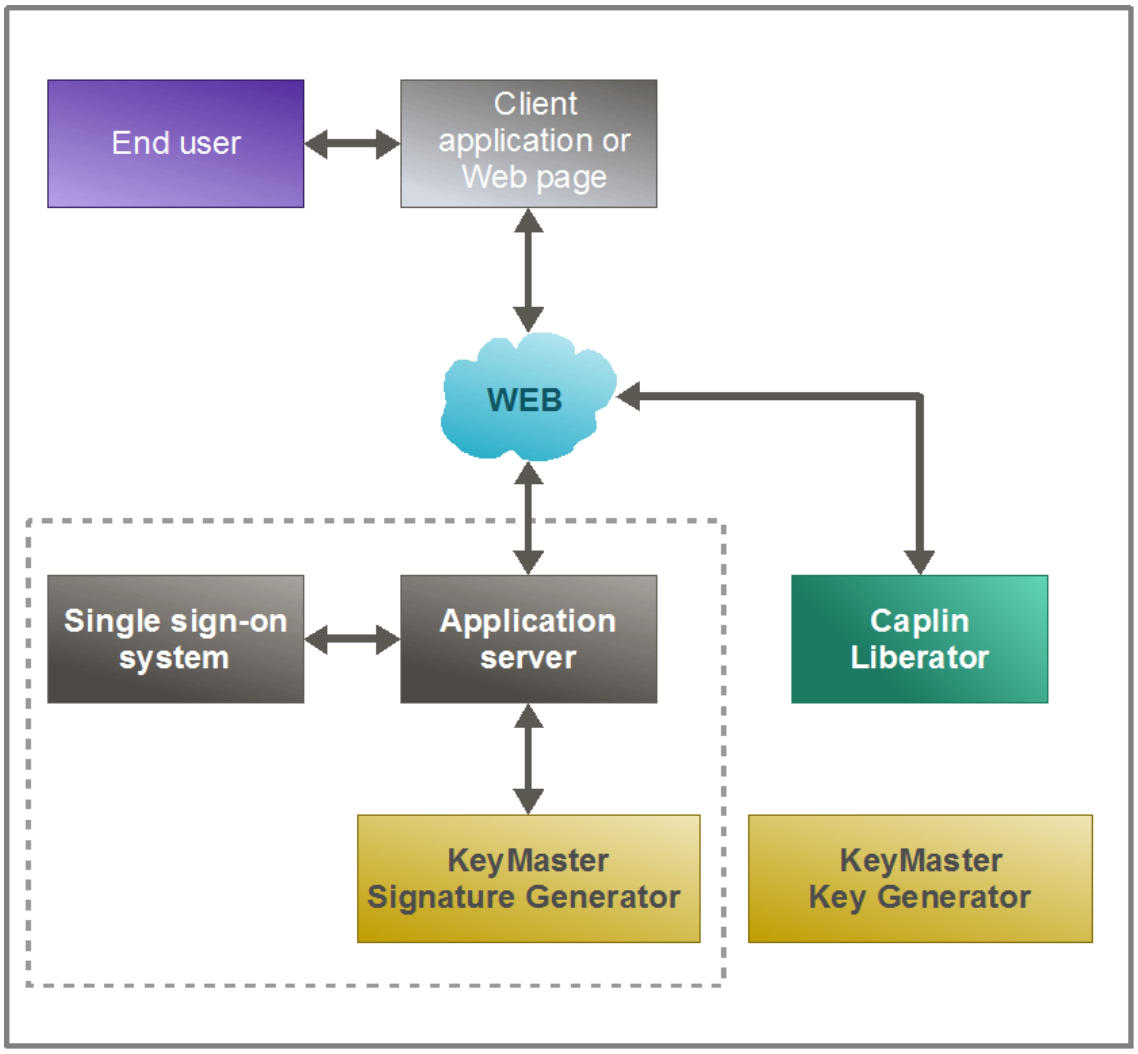
The token contains a digital signature which the Liberator can check to ensure that the token is a genuine one. The user is given access to the Liberator's data, provided the Liberator recognizes the user as valid and the credentials token is also successfully validated.

KeyMaster uses public key cryptography to generate the digital signature in the user credentials token; this is an extremely secure form of signature.

For a more detailed explanation of how these concepts are implemented, see the KeyMaster architecture 8 and How does KeyMaster work 11. Also see Examples of use 23, the note on public key cryptography and digital signatures 28 and the Glossary of Terms and Acronyms 31.

# 3 KeyMaster architecture

The following diagram shows the architecture of KeyMaster and how it fits in with the Caplin platform and third-party/customer software.




**KeyMaster Architecture**

KeyMaster works in conjunction with a Caplin Liberator server and an application server. The end user accesses the application server and the Liberator via a client application, or via web pages driven by Java, JavaScript, or Microsoft Silverlight.

## Application server

The application server is customer-supplied and may already be implemented as part of an existing web site infrastructure. In the context of KeyMaster the application server is responsible for primary authentication of the end user, for example through a login sequence requiring the user to supply a valid username and password.

The gray dashed line in the diagram bounds the components of the application server, including the single sign-on system and the KeyMaster Signature Generator.

## Single sign-on system

The single sign-on system is customer-supplied and may be an existing part of the web site infrastructure. It runs under the control of the application server and allows end users to sign on ("log in") just once to the server. The system authenticates the user, for example via a name and password entered at the user's workstation, or perhaps using more sophisticated technology such as a Smart Card. Successfully authenticated users can then access all the applications to which they have been given access rights, without needing to explicitly log in separately to each application.

## Caplin Liberator

The Caplin Liberator server provides real-time data and trading capability for the client application or Web pages, but only when it has successfully authenticated the end user and granted access to the relevant data. The authentication is performed using appropriate Caplin Auth Modules (XMLauth, cfgauth, and custom Auth Modules built using the javaauth or C Auth SDKs).

## KeyMaster

KeyMaster integrates Caplin Liberator with the single sign-on system, so that end users do not need to explicitly log in to a Liberator server in addition to their normal log in procedure. It uses public key encryption to produce digitally signed user credentials tokens that authenticate users for Liberator access. End users are not aware of these tokens and the process of logging in to the Liberator using the token is transparent to the users.

KeyMaster comprises two tools: a key generator and a signature generator.

- **Key Generator**

  The Key Generator is a Java application used to create an encryption key pair; one key is the private key and the other is the public key. KeyMaster uses the private key to sign the user credentials token that authenticates a user's access to the Liberator. The public key is exported to the data provider's Caplin Liberator for use during the authentication process.

  The KeyMaster Key Generator can be used to generate key pairs for use with the Java-based Signature Generator. Key pairs can also be generated using third-party tools, such as the OpenSSL key generation commands (see www.openssl.org); this is necessary if the Signature Generator is implemented using KeyMaster.NET, or if KeyMaster is to be integrated with a secure key storage hardware module.

- **Signature Generator**

  The Signature Generator runs under the control of the application server behind the single sign-on system. It creates the signed user credentials token; the token is digitally signed (encrypted) using the KeyMaster private key. A Liberator server can use this token to validate an end user's login to the server.

The Signature Generator is usually an application server module. A useable Signature Generator is provided with the Java-based KeyMaster product, as a Java servlet. You can customize this servlet as required, or you can use it to guide the design of a similar module in another technology. You can also use the KeyMaster.NET SDK to implement a Signature Generator that runs as a Microsoft .NET application (typically deployed as an ASP.NET web page) – see the **KeyMaster.NET API Reference**.

Also see the note on public key cryptography and digital signatures 28.

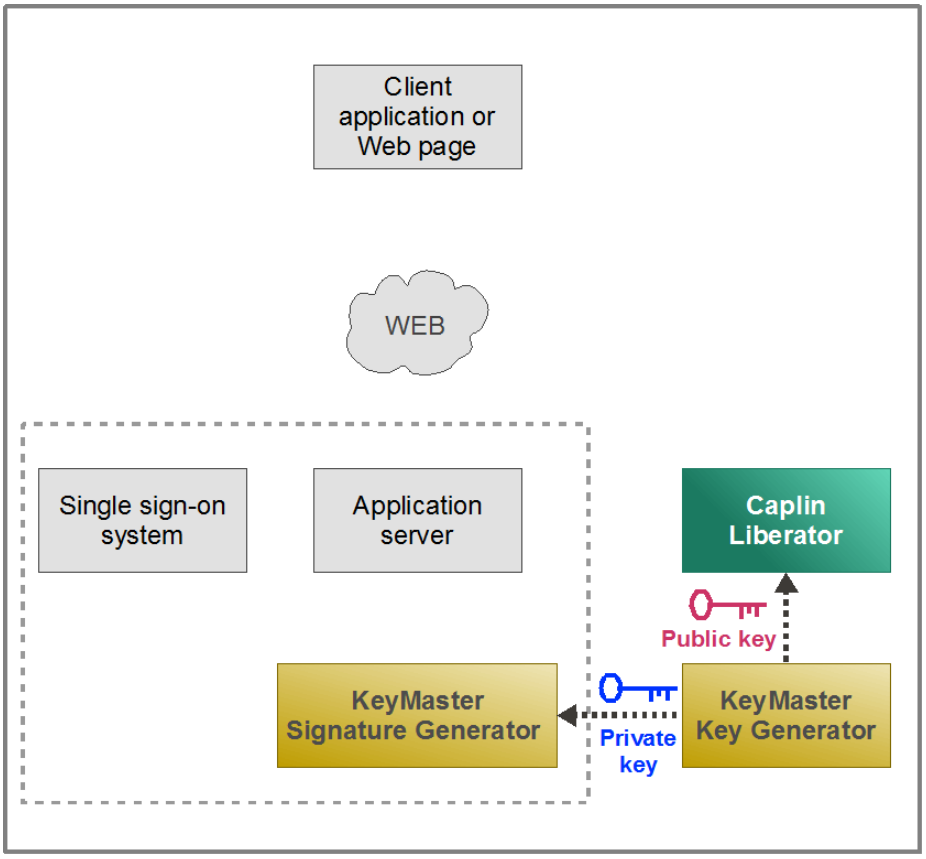
# 4 How does KeyMaster work?

Assume an end user has successfully logged in to an application server, via a client application or a Web page. As part of the user's work flow it is necessary to communicate with a Caplin Liberator server, and this requires that the user log in to the Liberator. Working in conjunction with a single sign-on system, KeyMaster allows the user to be authenticated on the Liberator and automatically logged in, with no manual intervention. The following diagrams show how this is achieved.

Note that the Liberator's authentication can be configured so that the end user can *only* log in to the Liberator via the application server and KeyMaster; the Liberator will reject any attempt by the end user to log in to it directly.

## 4.1 a) Generating the public/private key pair

Before KeyMaster can be used to authenticate user logins you must generate a public/private encryption key pair; this is a one-off task.




**Generating the key pair**

If you are using KeyMaster's Java-based Signature Generator, you can use the KeyMaster Key Generator (also Java-based) to generate the key pair, as shown in the previous diagram.

If the Signature Generator is implemented using KeyMaster.NET, or if KeyMaster is to be integrated with a secure key storage hardware module, you should generate the key pair using the relevant OpenSSL commands – see the **KeyMaster.NET API Reference** and the **KeyMaster Administration Guide** respectively.

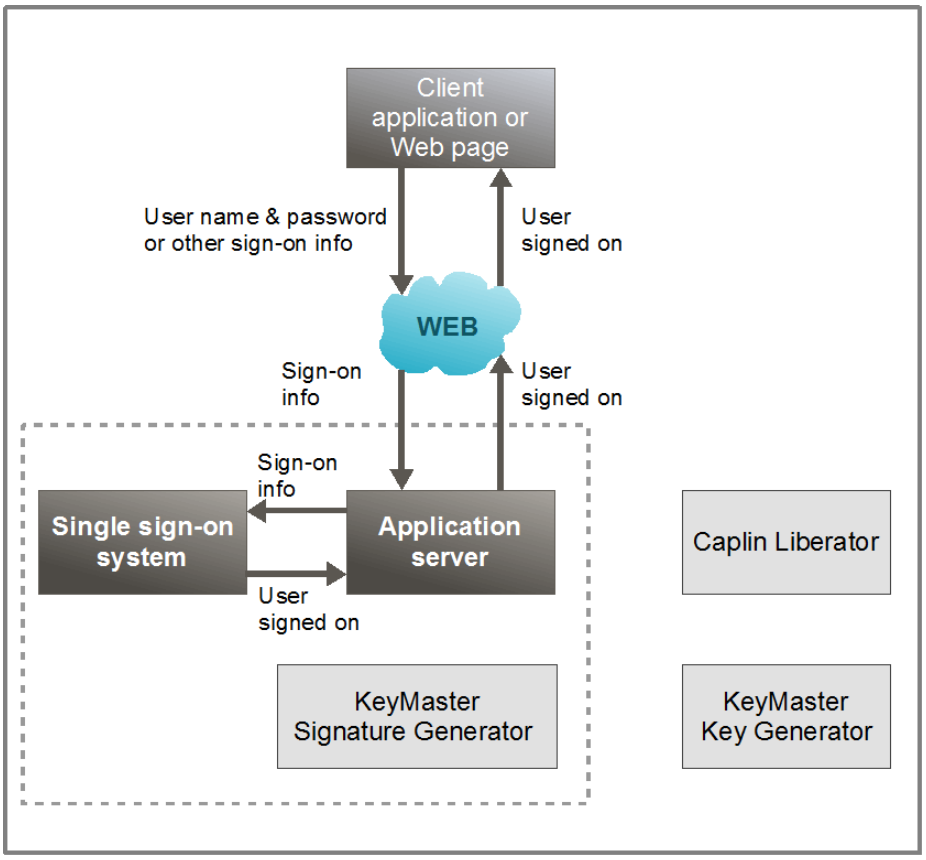
You then manually distribute the keys to the required locations; the private key is placed with the KeyMaster Signature Generator or secure key storage hardware module, and the corresponding public key is placed with the Liberator server (the Liberator actually uses a binary version of the public key, called a DER 31 key file).

## 4.2 b) User sign-on

Caplin KeyMaster is intended to be used with an existing single sign-on system for authenticating end users. Typically an end user runs a client application, or accesses a web page, that will subsequently display data provided by a Caplin Liberator. Before the client can access the Liberator, the end user must first sign on ("log in"), so that they can be authenticated.

The client captures the relevant sign-on information entered by the user; for example, a user name and password. It sends this information to the application server, which passes it to the single sign-on system. The single sign-on system authenticates the user, and, assuming the user is recognized, it informs the client (via the application server) that the user is signed on.

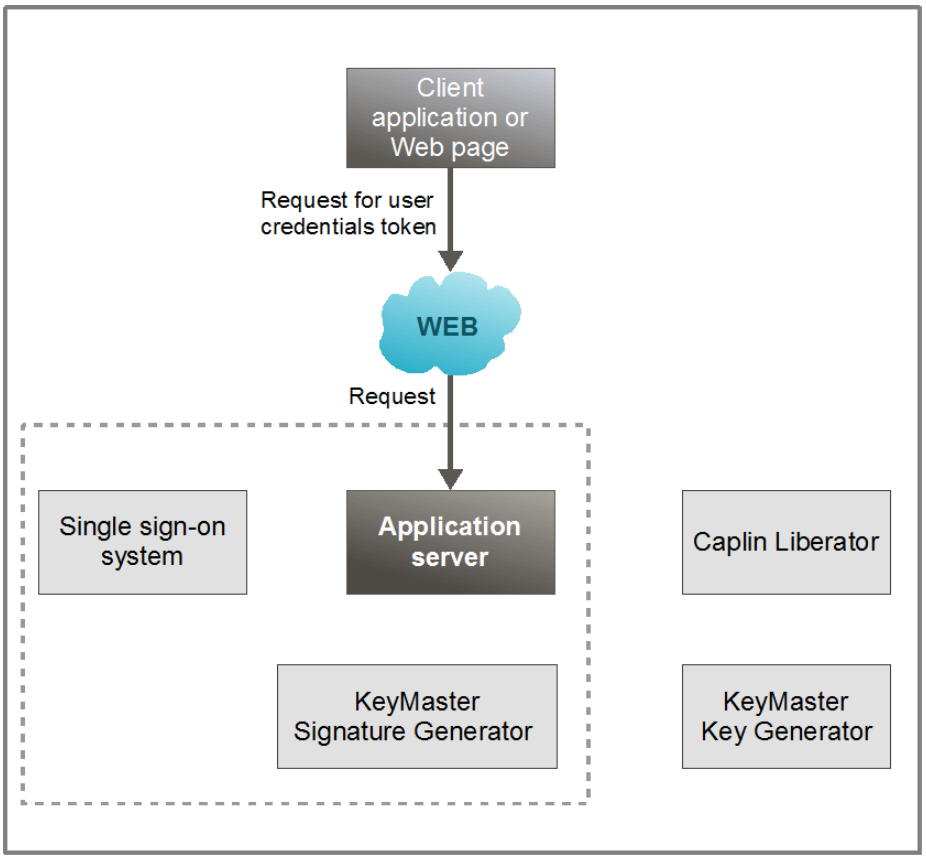
The sign on process does not involve any KeyMaster components.




**Single sign-on of end user**

## 4.3 c) Requesting the token

Subsequently, code within the end user's client application or Web page wishes to access the Liberator server. Rather than requiring the end user to explicitly log in to the Liberator, the code can make use of the fact that the end user has already signed on successfully. It therefore sends a request for a user credentials token to a predefined KeyMaster URL; the URL is known to the application server and when invoked it will create the token. The user credentials token will allow the client to log in to the Liberator directly, so that the end user does not have to enter any sign on information again.

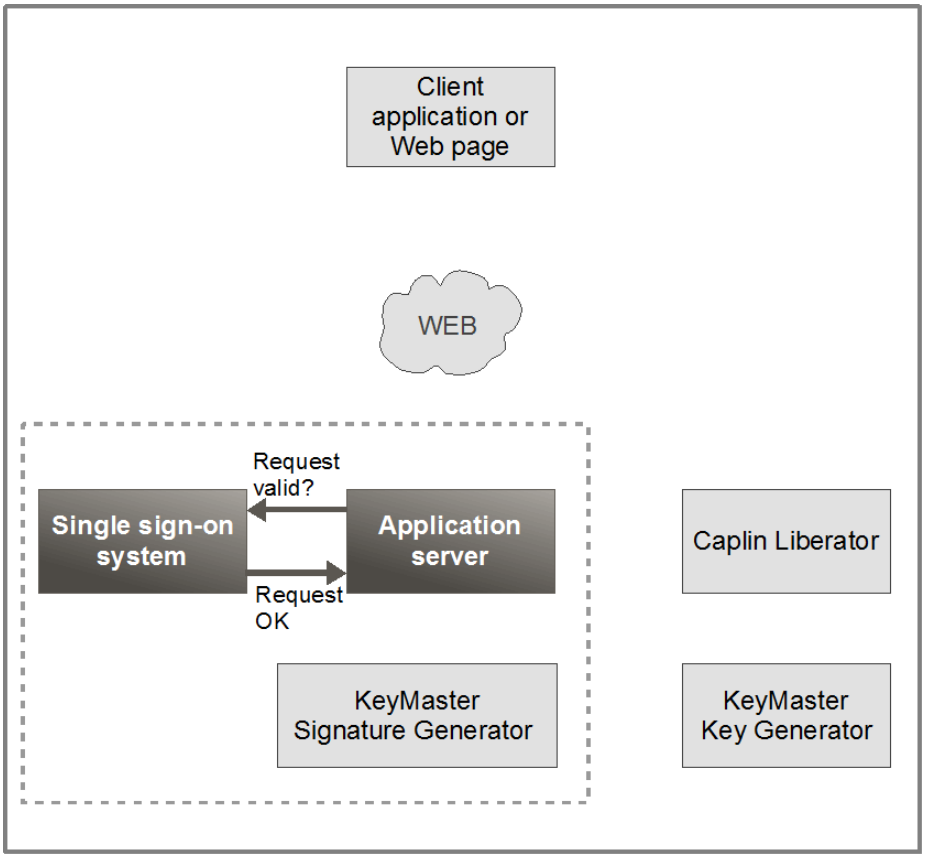



**Requesting the token**

The StreamLink code that sits behind the client application or web page automatically generates the URL request when it detects that the Liberator server needs to be accessed for the first time during the user's session. The required KeyMaster URL is defined in the application's StreamLink configuration.

## 4.4 d) Checking the user permissions

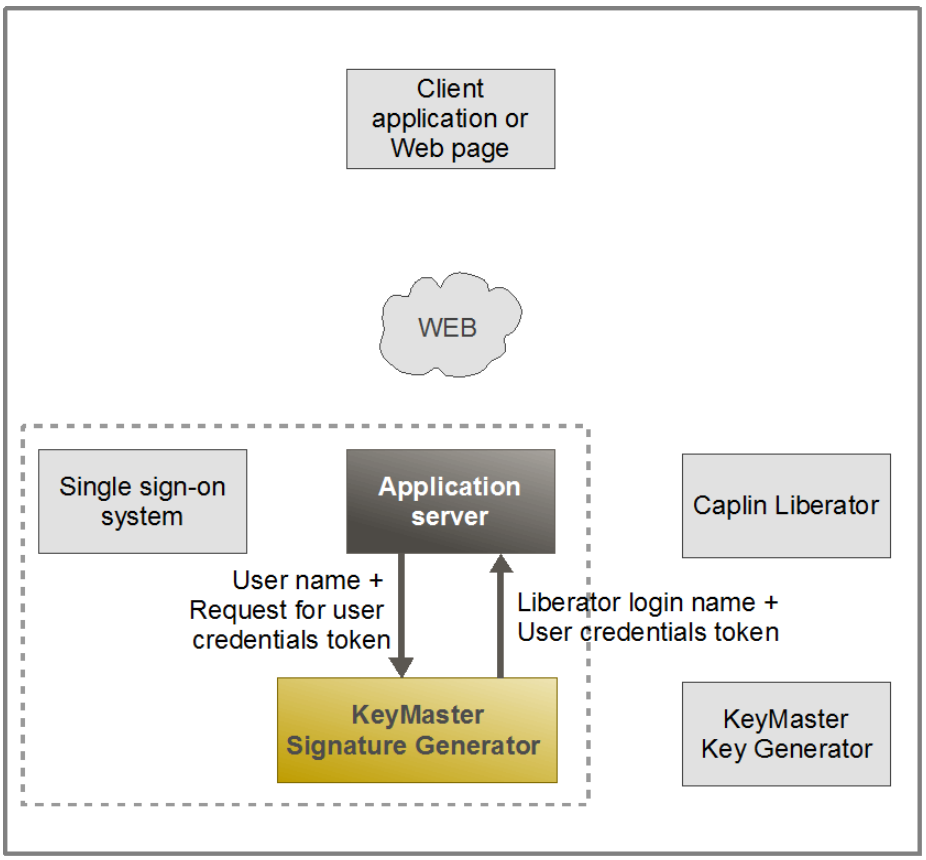
The application server asks the single sign-on system to check that the request for the user credentials token is valid. For example, is the user's request in the context of a valid session – that is, did the user previously log in successfully and is the session still in force (not timed out)?




**Checking the user permissions**

## 4.5 e) Creating the user credentials token

KeyMaster's Signature Generator now creates the requested user credentials token. The Signature Generator is secured behind the single-sign on system and so can only be invoked by a URL request that has been validated through the single sign-on. This prevents unauthorized users or applications from creating illegal tokens by calling the Signature Generator directly.




**Creating the user credentials token**

To generate the token, KeyMaster first creates a unique login identifier comprised of (at least) a date-time stamp and sequence number. It could also contain additional information, such as the user name. This identifier is then signed with a signature generated by KeyMaster's signature generator, to produce the user credentials token.

In standard KeyMaster the digital signature is generated using the **MD5withRSA** hashing algorithm, which incorporates RSA® encryption 32, a form of public key encryption (see A note on public key cryptography and digital signatures 28 and in particular Digital signature algorithms 29). A wide variety of other digital signature algorithms are also supported; see Supported digital signature algorithms 29.

The signature consists of the login identifier, encrypted using the private key. Since this encryption key is not known to the outside world, no one else can independently produce an equivalent user credentials token with a valid signature. Any attempt by a third party to subsequently modify the token will invalidate it, in the sense that when the signature is decrypted it will not match the unencrypted login identifier in the token, thus revealing that the token may have been tampered with (see Logging in to the data provider's Liberator server 18).

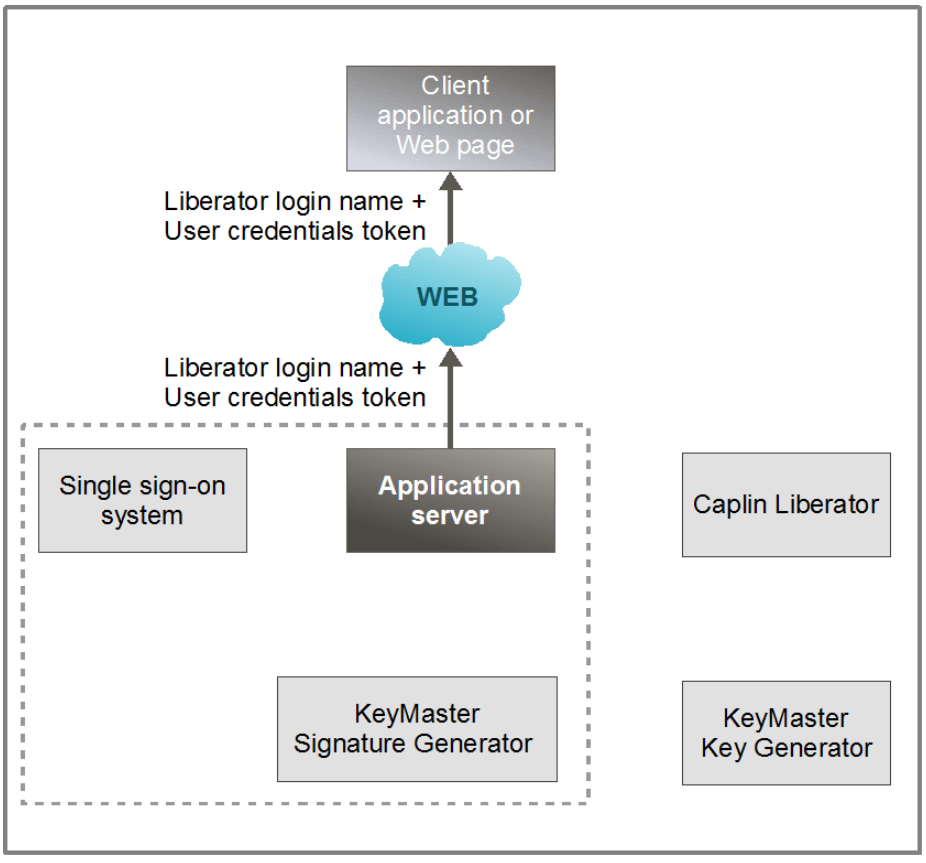
Note that in the standard KeyMaster implementation the data within the token, such as the login identifier, is also sent as plain text within the token.

KeyMaster passes back to the application server the user credentials token and a Liberator login name; these will subsequently be used to log in to the Liberator server. The Liberator login name can be the same as the user name of the end user whose client application is requesting access to the Liberator. However, it could instead be a generic login name allocated to a particular class of end users who have access to a particular Liberator service. In this situation the Liberator does not need to validate the individual end users who request the service, since this has already been done through the single sign-on. It just uses the user credentials token to validate the service request.

If a token cannot be created, for example if the user has no access, a message is returned to the client giving details of the problem.

## 4.6 f) Passing the token to the client

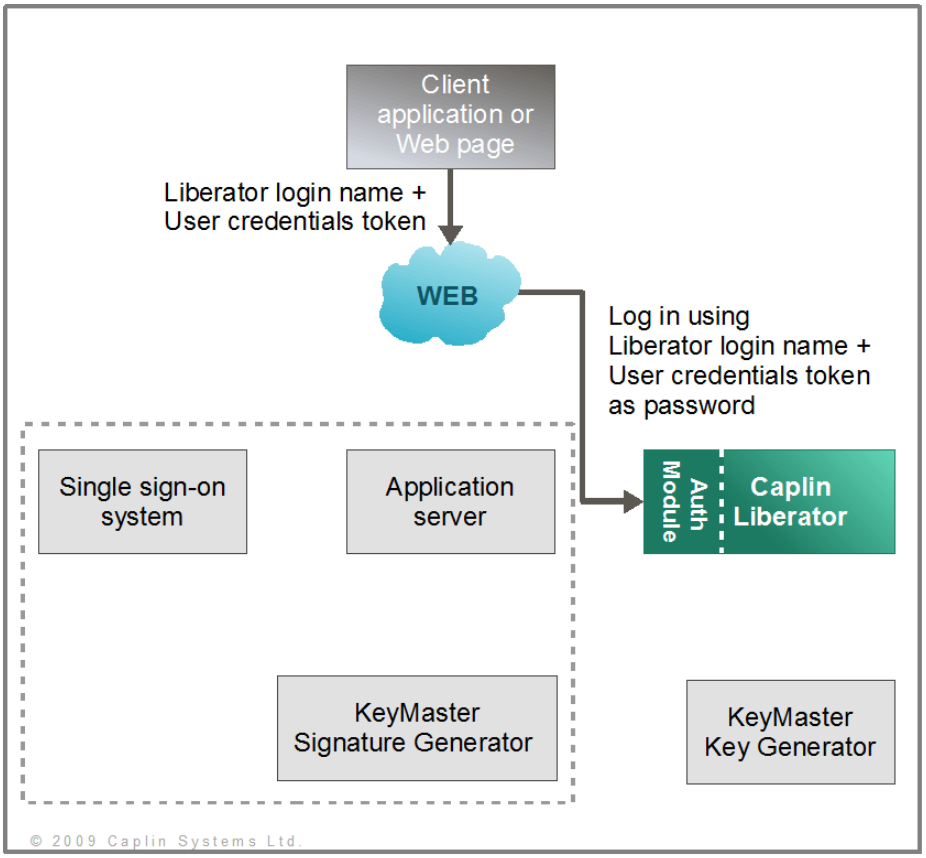
The application server returns the Liberator login name and user credentials token to the client application or web page. After this, the application server plays no further part in the authentication of the end user on the Liberator.




**Passing the user credentials token to the client**

## 4.7 g) Logging in to the data provider's Liberator server

The Liberator login name and user credentials token enable the client application or web page to log in to the Liberator on the end user's behalf, as follows.




**Logging into the data provider's Liberator server**

The application or web page sends the Liberator login name and the user credentials token to the Liberator. The Liberator uses this information to authenticate the login request. It identifies the Liberator user corresponding to the login name and obtains the public key assigned to that user (this assignment is determined by the Liberator configuration – for details see the **KeyMaster Administration Guide)**.

The Liberator's authorization module (Auth Module) uses the public key to decrypt the signature in the user credentials token. If the decrypted signature matches the date-time stamp and sequence number in the token, then the login request is authentic and the login is successful. If the decrypted signature does not match, then the user credentials token is either not genuine or has been tampered with, so the Liberator rejects the attempt to log in.

Tokens can only be used once and have a limited life. The Liberator will reject attempts to reuse tokens; it does this by checking the token's sequence number (has the token been previously submitted?), and by checking the token's time stamp (has the token timed out?).

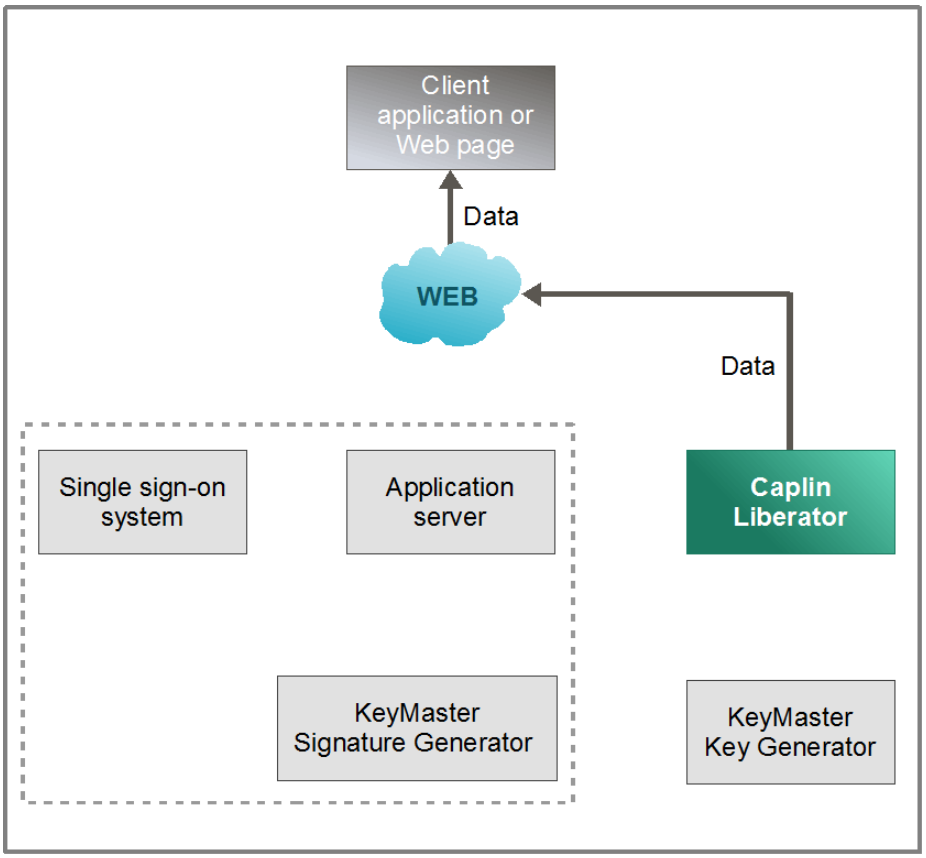
Access to the Liberator server is secure because the user credentials token that is used to log in to the Liberator has been created and signed by an instance of the KeyMaster Signature Generator. The Signature Generator is protected behind the single-sign on system which authenticated the end user. The Liberator can therefore safely employ the user credentials token to authenticate the user; in effect it is

using the single sign-on facility, but independently of the particular implementation of the single sign-on system that has been deployed.

For added security KeyMaster can be configured so that the user credentials token contains the end user's login name. This name is also digitally signed in the token. The Liberator's Auth Module can therefore check that the login name in the token is valid, and can compare it against the login name supplied by the client application. Only if the three copies of the login name match will the Liberator authenticate the login request.

## 4.8 h) Receiving data from the data provider's Liberator server

If the login to the Liberator was user was successful then the Liberator can send data to the application, and exchange trade messages with it, via the RTTP protocol.




**Receiving data from the data provider's Liberator server**

# 5 Other Caplin Xaqua components

KeyMaster is used in conjunction with various other Caplin Xaqua components:

- Server side components
- Client side components

## 5.1 Server side components

Before a Caplin Liberator server can supply data to a user application for the first time, it must authenticate the user and determine the user's access permissions. This includes the situation where the user is to be authenticated through a KeyMaster user credentials token. The Liberator performs the authentication and grants access permissions via an authorization module. There are several flavors of authorization module:

- **XMLauth**

  XMLauth is supplied as standard with Liberator. It can easily be configured to authenticate users who are identified via a KeyMaster user credentials token. For details of how to do this refer to the **KeyMaster Administration Guide**.

- **cfgauth**

  cfgauth is also supplied with Liberator. This authorization module is intended for relatively low numbers of users where the usernames and other details do not need to be changed often. It can easily be configured to authenticate users who are identified via a KeyMaster users credentials token. For details of how to do this refer to the **KeyMaster Administration Guide**.

- **JavaAuth API**

  You can develop your own Liberator authentication module in Java using the **Liberator JavaAuth API** (javaauth). This module can be implemented to authenticate users via KeyMaster user credentials tokens.

- **C Auth API**

  You can develop your own Liberator authentication module in C using the **Liberator Authentication C API**. The Authentication API provides a mechanism for validating KeyMaster user credentials tokens using the **signature_check()** function.

## 5.2 Client side components

A client side application interacts with KeyMaster to request a user credentials token, and it subsequently sends the token to any Liberator servers with which it needs to interact. The client code performs these actions with the help of functionality provided by the client side Caplin StreamLink libraries:

| Integrate KeyMaster authentication into: | Associated StreamLink Library: | StreamLink documentation: |
|---|---|---|
| Web client applications that use StreamLink for Java. | StreamLink for Java | **StreamLink for Java API Reference** |
| Web pages that use StreamLink for Browsers. | StreamLink for Browsers | **StreamLink for Browsers API Reference** |
| .NET client applications. | StreamLink .NET | **StreamLink.NET API Reference** |
| Web pages that use StreamLink for Silverlight. | StreamLink for Silverlight | **StreamLink for Silverlight API Reference** |

# 6 Java customization

The KeyMaster for Java distribution kit contains a standard version of the product. This can configured by modifying various configuration files (see the **KeyMaster Administration Guide**). However, in many instances KeyMaster will need to be integrated into an existing single sign-on system, and this may require some product customization.

The Java KeyMaster can be customized in various ways.

For example:

- Encryption package.

  The standard version of Java KeyMaster is shipped with public domain encryption software from The Legion Of The Bouncy Castle. You can replace this with some other JCE 31 provider's Java class.

- User credentials token

  The user credentials token produced by standard KeyMaster consists of a date-time stamp, a sequence number, the user name (if configured), and the digital signature of these items. You can customize KeyMaster to add additional information to the token. For example, you could add and sign some simple information about what sort of Liberator service the end user is entitled to, such as a news wire channel number.

- Response to request for a user credentials token

  Standard KeyMaster is shipped with Java classes that will supply user credentials tokens to Javascript applications and to Caplin StreamLink applications. These are examples of **ResponseFormatter** classes; these classes format KeyMaster's response to a client application's request for a user credentials token. You can add custom **ResponseFormatter** classes that format responses more appropriately to the needs of your client applications.

For more information on the public KeyMaster Java classes that can be called and extended to produce customized versions of KeyMaster, see the **KeyMaster Java API Reference**.

# 7 .NET customization

The .NET version of KeyMaster is supplied as an SDK with an API that is documented in the **KeyMaster API Reference**. You create a KeyMaster Signature Generator by writing your own implementations of the following .NET interfaces defined in the KeyMaster.NET API:

- **IAuthenticationParameters**
- **IKeyMasterConfiguration**
- **IKeyMasterFormatter**.

The API also includes some example implementations of these interfaces and a main class (**KeyMaster**).

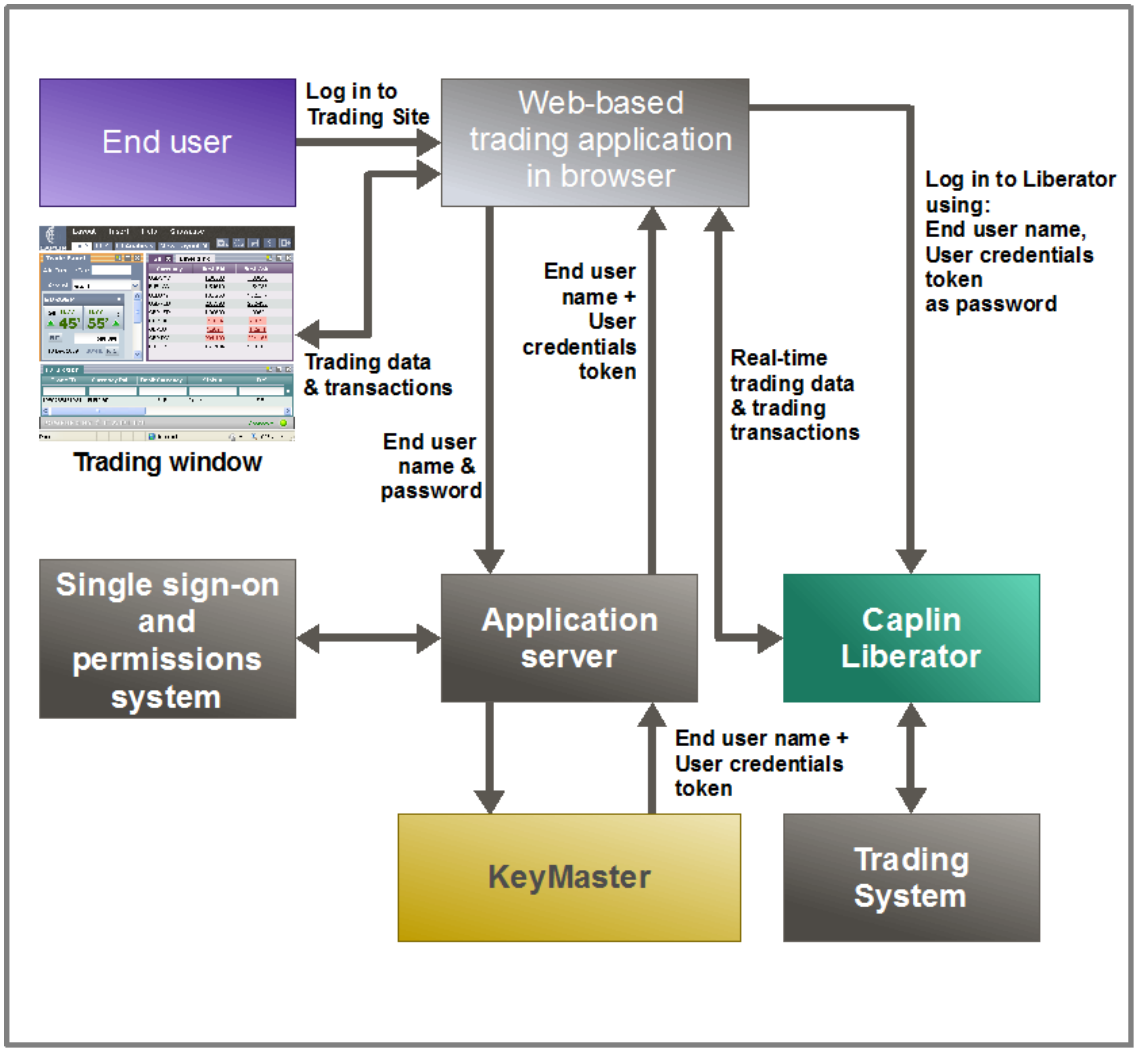
For more information, see the **KeyMaster API Reference**.

# 8 Examples of use

Here are three business scenarios where KeyMaster is used to support a single sign-on capability.

## 8.1 Using KeyMaster in a web-based trading application

This scenario shows how KeyMaster can be used in a web-based application for trading financial instruments.




**Using KeyMaster in a web-based trading application**

The trading application runs on the end user's desktop computer. It is loaded from a web application server, but the actual trading interaction is carried out with a Liberator server. So that the trading data and transactions do not have to be relayed through the application server, the trading application on the desktop obtains trading data directly from the Liberator, and sends trading transactions directly to the Liberator, which then passes them on to a dedicated trading system.

KeyMaster is integrated with a single sign-on and permissions system, so that every end user who trades is uniquely identified throughout the system.

To use the trading application, an end user visits a page on the trading web site and logs in to the service. The web application server authenticates the user, via the single sign-on and permissions system. If the user is successfully authenticated, KeyMaster is called to generate a user credentials token. The application server then sends the user credentials token to the user's instance of the web-based trading application, together with the user name.

When the web-based trading application needs to access the trading data, it logs in to the Liberator server using the user name and the user credentials token.

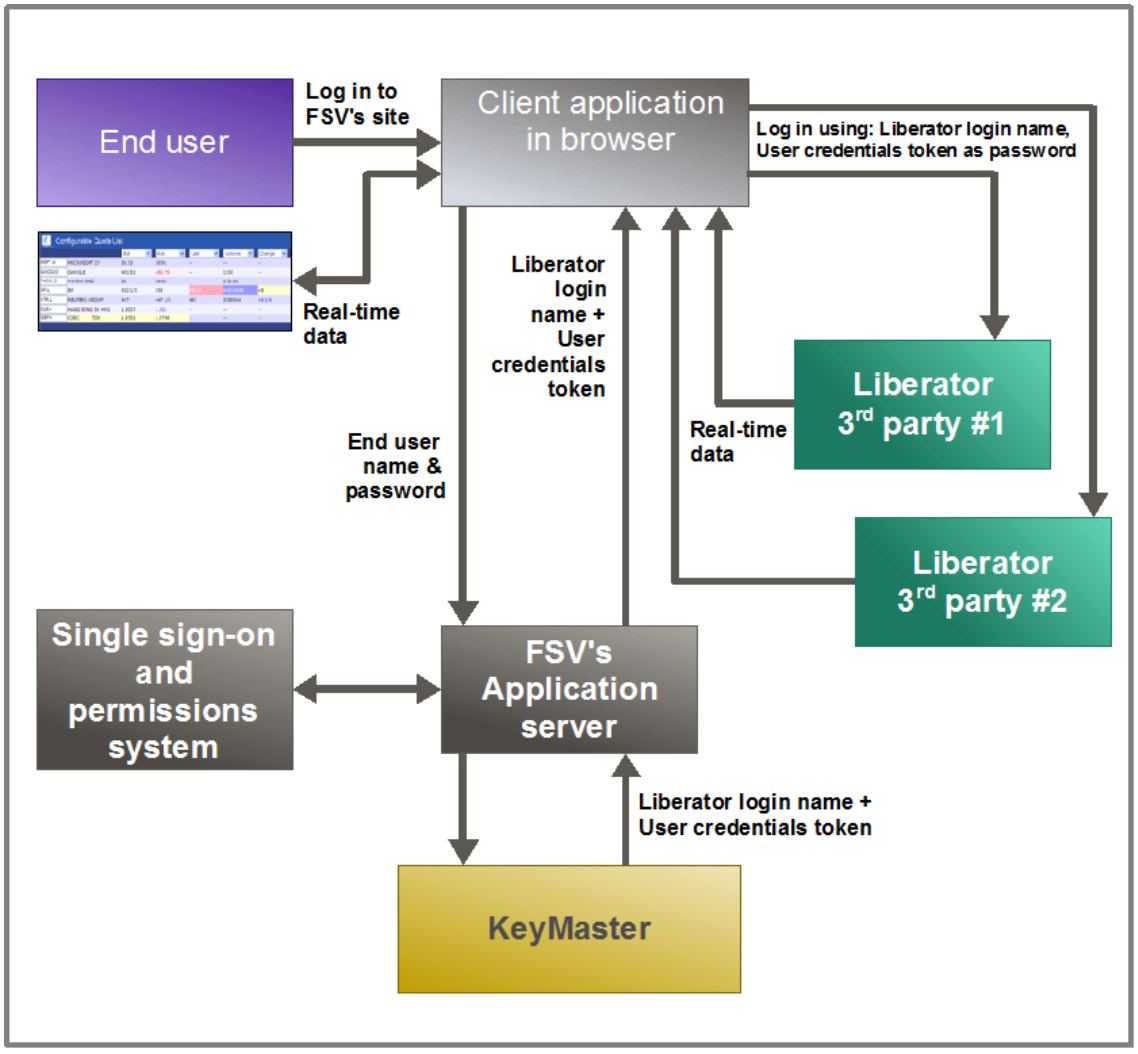
KeyMaster makes the process of logging in to the Liberator servers transparent to end users; they log in to the trading application just once and can then start trading.

Access to the Liberator server is secure because:

- The user credentials token that is used to log in to the Liberator has been created and signed by an instance of KeyMaster. This KeyMaster instance is under the control of the web application server which authenticated the end user.
- For added security KeyMaster can be configured so that the user credentials token contains the end user's login name. This name is also digitally signed in the token. The Liberator's Auth Module can therefore check that the login name in the token is valid, and can compare it against the login name supplied by the client application. Only if the three copies of the login name match will the Liberator authenticate the login request.

## 8.2 Accessing data feeds from multiple sources

In this scenario a financial services vendor (FSV) offers web users a real time data feed service, where the data feed is an aggregate of data obtained from two 3rd party vendors. The 3rd party vendors supply their data via Caplin Liberator servers.




**Accessing data feeds from multiple sources**

The data from the Liberator servers is aggregated by a client application that has been loaded into the user's browser from the FSV's web site. So that the real-time data does not have to be relayed through the FSV'S web site, the client application obtains the data feeds directly from the 3rd party vendors.

To use the data feed service, an end user visits a page on the FSV's web site and logs in to the service. The FSV's web application server authenticates the user, via its user sign-on and permissions system. The permissions system determines which 3rd party vendors the user may access, and uses KeyMaster to generate a user credentials token for each of the permitted Liberator servers. The FSV's application server then sends the user credentials tokens to the user's client application, together with the relevant Liberator login names.

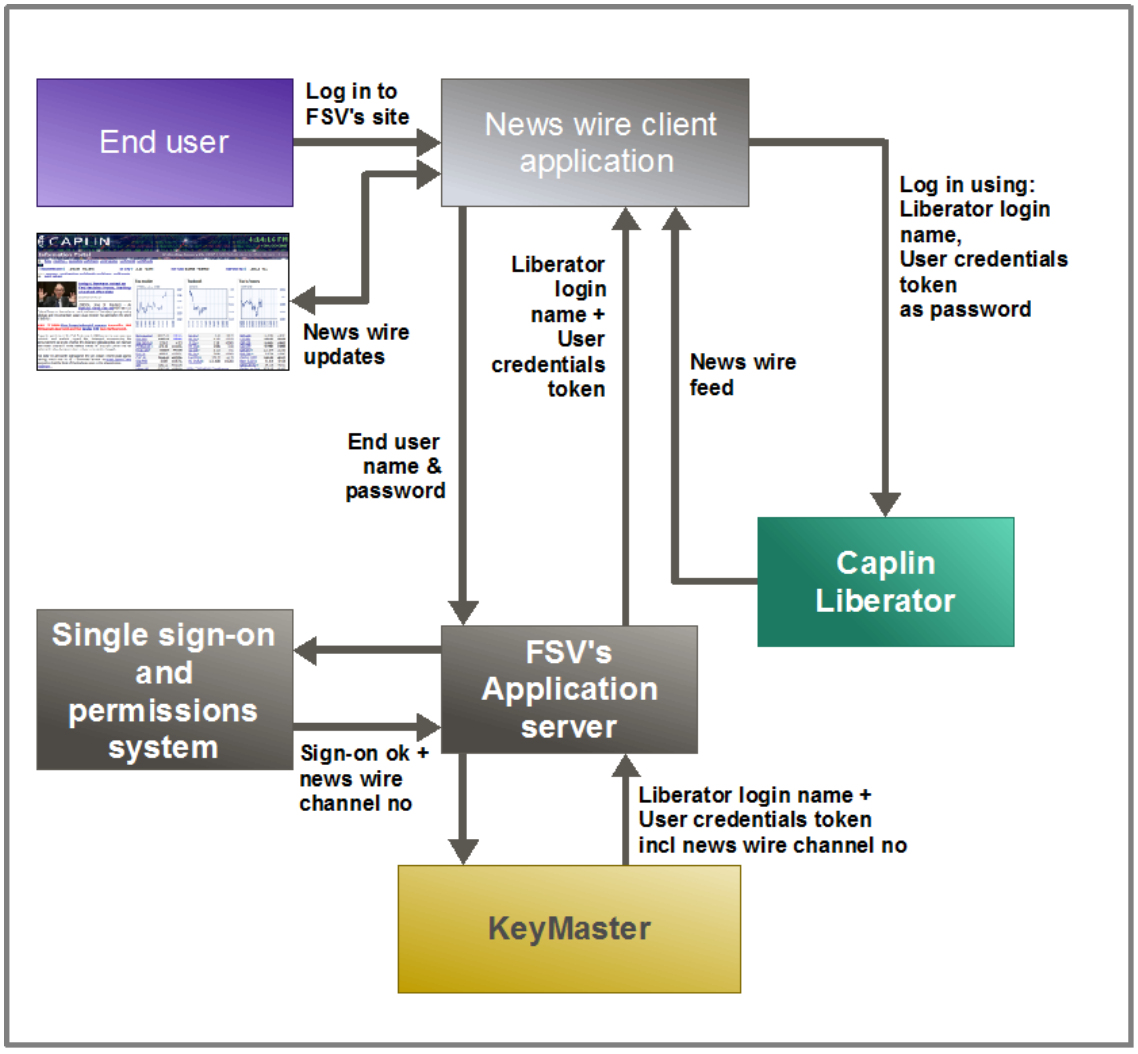
The client application logs in to each required Liberator server using the Liberator login name and relevant user credentials token. If the logins are successful, each Liberator will send streaming data via RTTP to the client application. The client combines the data from the two feeds seamlessly into a single display.

By using KeyMaster the process of logging in to the Liberator servers is transparent to end users; they log in to the FSV's web site just once and start receiving the aggregated data.

Note that because the data feeds are actually obtained from Liberator servers owned by 3rd parties, the end user's name is not required to log in to either of these servers. Instead, when the FSV's user sign-on system has authenticated and authorized the user, it allocates a Liberator login name for each of the 3rd party Liberator servers. These Liberator login names are generic; each login is configured on the Liberator to allow access to the particular types of real time financial data which the 3rd party vendor is contracted to supply to the FSV (for example UK Equity prices, Bond prices, and so on).

## 8.3 Accessing data feeds by user entitlement

In this scenario a financial service vendor (FSV) offers web users a real time news wire service, where a Caplin Liberator server supplies the news feed direct to the user's client application.




**Accessing data feeds by user entitlement**

End users have subscriptions that entitle them to access particular news wire channels. To access the service, an end user logs in to the FSV's web application server and the server authenticates the user via its user sign-on and permissions system. The permissions system determines which news wire channel the user can access, and then the application server uses KeyMaster to generate a user credentials token. KeyMaster has been customized to include the news wire channel number in the user credentials token.

The FSV's web server sends the user credentials token to the user's client application, together with a generic Liberator login name. The client application logs in to the Liberator server using the supplied Liberator login name and the user credentials token. The Liberator's Auth Module 31 has been customized to read the news wire channel information in the user credentials token; it uses this information to determine the real-time news wire data that will be sent to the end user's client application.

# 9 A note on public key cryptography and digital signatures

This is an explanation of the principles behind public key cryptography and digital signatures, and how they are used in KeyMaster.

## 9.1 The public key concept

Public key cryptography is a method of sending encrypted information between two parties without the need for them to exchange a key for encrypting and decrypting the information. Rather than using a single key it uses two related keys – a public key and a private key.

Public key cryptography can also be used to create and verify digital signatures.

KeyMaster uses RSA® encryption 32, which is a form of public key encryption.

## 9.2 Encryption

Assume Alice wishes to send Bob an encrypted message using public key cryptography. Bob has previously told Alice his public key, which may be used to encrypt messages to be sent to him. The key is called a public key because Bob has made it freely available to anyone who wants to use it, including Alice.

Alice uses Bob's public key to encrypt her message, and then sends it to Bob. Now if someone intercepts the message they cannot decrypt it. This is because Bob's public key only allows encryption – it cannot be used to decrypt messages; not even Alice can decrypt the encrypted version of her message to Bob.

Bob has a *private* key, corresponding to his public key. It is private because it is only known to him and he never reveals it to anyone else. When Bob receives the message he uses his private key to decrypt it. Since only Bob knows his private key he is the only person who can decrypt messages that have been encrypted using his public key.

## 9.3 Digital signatures

Public key cryptography can be used to digitally sign messages or documents in a secure way.

Assume Alice wants to sign her message to Bob, so that he will know it really did come from her. Alice also has a public key and private key (they are different to Bob's public and private keys of course). She puts her name in the message to Bob, and, before using Bob's public key to encrypt the message, she encrypts her name *using her private key*.

When Bob decrypts the message with his private key he sees that it contains an encrypted signature. Since he knows the message purports to come from Alice he decrypts the signature using Alice's public key. Now Alice's public key will only successfully decrypt text that has been encrypted using her private key, and no one but Alice has access to her private key. So, if when the signature is decrypted it reveals Alice's name, Bob knows that the message really did come from Alice.

## 9.4 Digital signature algorithms

A simple way to digitally sign the user credentials token would be to use RSA to encrypt all the data in the token using the KeyMaster private key. The resulting signed token contains both the signature and the original unencrypted data. The recipient of the token uses the KeyMaster public key to decrypt the signature and then compares this with the unencrypted data. If they match, the token is genuine – it really was generated by KeyMaster and has not been tampered with during transmission.

However encrypting and decrypting all the data using RSA is computationally expensive, particularly if the token contains a large amount of data. Therefore KeyMaster signs user credentials tokens using an algorithm that combines RSA encryption with a cryptographic hash function, since this is more efficient. A number of such algorithms are available (see “Supported digital algorithms” below).

By default, KeyMaster uses the MD5withRSA algorithm, which works like this:

1. Using the MD5 algorithm, compute a smaller hashed version of the data in the token; this hashed version is called the “message digest”. MD5 is effectively a one-way hashing algorithm, since it is very difficult to deduce the original text from the message digest.
2. Use the KeyMaster private key to encrypt only the message digest.

This procedure is computationally inexpensive.

The signed token contains the digitally signed message digest (the “signature”) and the original unencrypted data. The recipient of the token (Liberator's Auth Module) validates it in the following way:

1. Use MD5 to produce a message digest of the unencrypted data in the token.
2. Decrypt the digital signature using the KeyMaster public key. The decrypted value is the original message digest.
3. Compare the two message digests. If they are the same, the token is genuine.

### Supported digital signature algorithms

You can configure KeyMaster to use one of following algorithms for digitally signing user credentials tokens:

- MD5withRSA* (default)
- RipeMD160
- SHA1
- SHA256*
- SHA384
- SHA512

SHA (as in SHA1, SHA256, and so on) stands for Secure Hash Algorithm; the SHA family of hash functions are a U.S. Federal Information Processing Standard.

**Note:** Only the algorithms marked * can be used with the Java-based KeyMaster Signature Generator. All the algorithms can be used in KeyMaster.NET, and in Liberator Auth Modules for Liberator versions 4.5.13 and above.

**Note:** **MD5 limitations:** Since KeyMaster was first released, the cryptographic community have found that the MD5 algorithm can produce hash collisions. This potentially compromises the algorithm.

Caplin has retained MD5withRSA as the default digital signature algorithm for backward compatibility with previous versions of KeyMaster. *However, customers installing KeyMaster for the first time may wish to configure the software to use a more secure algorithm, such as SHA256.*

# 10 Glossary of Terms and Acronyms

This section contains a glossary of terms, abbreviations, and acronyms relating to the KeyMaster product.

| Term | Definition |
|---|---|
| **ASP** | Active Server Pages.<br>A technology from Microsoft that dynamically generates web pages using server-side scripts. Also known as “Classic ASP”.<br>Also see **ASP.NET**. |
| **ASP.NET** | A newer version of Microsoft's Active Server Pages (**ASP**) that dynamically generates web pages using **.NET** technology. |
| **Authentication** | In the context of KeyMaster, authentication is the process of identifying a user, for example by checking a user name and password that the user supplied when attempting to log in.<br>Authentication must proceed **authorization**. |
| **Auth Module** | A Caplin Xaqua software module that performs **authentication** and **authorization** functions.<br>The Liberator server uses Auth Modules to authenticate users who log in to the Liberator, and then to determine the users' access permissions to Liberator objects. |
| **Authorization** | In the context of KeyMaster, authorization is the process of determining the access rights that a user has to resources, such as data and functionality provided by computer software.<br>Users cannot be authorized until they have been successfully authenticated – see **authentication**. |
| **Caplin Liberator** | Caplin Liberator is a real-time financial internet hub that delivers trade messages and market data to and from subscribers over any network. |
| **DER** | Distinguished Encoding Rules.<br>Rules for encoding ASN.1 objects in binary format which define just one way to represent any ASN.1 value. DER encoding is typically used when the same object is encoded in ASN.1 format multiple times for digital signature verification. |
| **DER public key file** | In KeyMaster this is a file containing a KeyMaster public key in **DER** format. This file is used by Caplin Liberator to authenticate the **user credentials token** sent by a user application that wishes to log in to the Liberator. |
| **Digital signature** | An electronic signature that is used to authenticate the sender of a message or author of a document. The signature is usually encrypted in some manner (see **public key encryption**).<br>KeyMaster inserts a digital signature in the **user credentials tokens** that it generates. |
| **JCE** | Java Cryptography Extension<br>A Java package that provides a framework for and implementations of encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms.<br>For more information see Sun's Java Cryptography Extension Reference Guide |

| | |
|---|---|
| **MD5withRSA** | See Digital signature algorithms 29. |
| **.NET** | A Microsoft framework for developing distributed applications that run under Microsoft Windows® operating systems and can easily intercommunicate with applications and systems running on different operating systems. |
| **OpenSSL** | An open source implementation of the SSL (Secure Sockets Layer) and TLS (Transport Layer Security) protocols. It includes commands for generating RSA key pairs and certificates.<br>See www.openssl.org. |
| **Public key encryption** | A method of sending encrypted information between two parties without the need for them to exchange a key for encrypting and decrypting the information. Rather than using a single key it uses two related keys – a public key and a private key.<br>See A note on public key cryptography and digital signatures 28. |
| **RSA® encryption** | RSA is a **public key encryption** algorithm developed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman. |
| **RTTP** | Real Time Text Protocol<br>Caplin's protocol for streaming real-time financial data from Liberator servers to client applications, and for transmitting trade messages between clients and Liberator in both directions. |
| **SDK** | Software Development Kit |
| **SHA256withRSA** | See Digital signature algorithms 29 |
| **SL4B** | See **StreamLink for Browsers**. |
| **Single sign-on** | A user authentication process in which a user supplies just one set of **user credentials** (such as a user name and password). The user can then access multiple applications and systems without being prompted for credentials again. |
| **StreamLink** | The StreamLink libraries connect client applications to **Caplin Liberator** via the **RTTP** protocol. They provide an object oriented API, on top of RTTP, that provides access to RTTP functionality. |
| **StreamLink for Browsers** | StreamLink for Browsers is a JavaScript implementation of **StreamLink** that runs in Web browsers. It allows Ajax applications to communicate with **Caplin Liberator**. |
| **StreamLink for Java** | StreamLink for Java is a Java™ implementation of **StreamLink**. It allows Java applications to communicate with **Caplin Liberator**. |
| **StreamLink.NET** | StreamLink.NET is a Microsoft .NET implementation of **StreamLink**. It allows .NET applications to communicate with **Caplin Liberator**. |
| **StreamLink for Silverlight** | StreamLink.NET is a Microsoft Silverlight™ implementation of **StreamLink**. It allows Silverlight applications to communicate with **Caplin Liberator**. |
| **User credentials** | Information used to authenticate a user; for example, a user name and password. |
| **User credentials token** | A data structure, containing **user credentials**, that is passed from one application to another in order to authenticate the user. |

## Contact Us

Caplin Systems Ltd
Triton Court
14 Finsbury Square
London EC2A 1BR
Telephone: +44 20 7826 9600
Fax: +44 20 7826 9610
**www.caplin.com**