1 About this document
The Dutch Pathology Society has decided to implement a national Pathology Image Exchange (PIE) solution, starting during the spring 2017. Initially 10 labs will be connected and grow up to the majority of Pathology labs in Holland. The technical solution for PIE is cloud based and built on standards such as DICOM, XDS and follows relevant IHE profiles. Since these standards are not fully implemented in digital pathology yet, the solution also supports proprietary file formats and ways of integrating with LIS and IMS systems.

This document describes the interface for launching the Manual Uploader, the Central Viewer and the Panel Module.

2 Description

![Diagram of PIE URL Launch API](image)
The user launches PIE from the LIS, dependant on the action that they wish to fulfil, Upload Slides, View Slides or enter the Panel Module. The launch of PIE will be via a URL launch initiated from the LIS, providing key information such as the action they wish to perform, the User ID, the current PIE case key, as specified by Lab2Lab, the local URL for the Patient Data Store and a local Site Key.

If the parameters have been encrypted (PKI) by the LIS then PIE will consider this a secure launch of PIE, and the launching user will be handed off to the appropriate sub system automatically without the need to directly log into PIE. If the parameters have not been encrypted, then the user will be expected to log into PIE before being handed off to the appropriate sub system.

Any user that is launching PIE for the first time will be asked to register for access to PIE, and will automatically be provided with a one-time password. All access to PIE will be maintained by a local administrator who is able to approve or deny access for their Laboratory.

3 Supported events

3.1 LIS -> PIE

The following are the supported events coming from LIS/IMS to PIE.

<table>
<thead>
<tr>
<th>Event name</th>
<th>Description</th>
<th>Consequences in PIE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Launch Manual Uploader</td>
<td>Image file and metadata file to be manually uploaded to the PIE Image Viewer</td>
<td>The IEP Interface will be launched allowing Pathology slides to be uploaded manually. The Relevant case key and a link to the local Data repository need to be supplied to ensure that the Slides are made available and can be viewed once they have been uploaded.</td>
</tr>
<tr>
<td>Launch Viewer</td>
<td>PIE Image Viewer will display images</td>
<td>The user will be authenticated by IEP and UniView will be launched to show the images related to the case key.</td>
</tr>
<tr>
<td>Launch Panel Module</td>
<td>Panel Module launched</td>
<td>The user will be authenticated by IEP and the Panel Module will be launched in user context.</td>
</tr>
</tbody>
</table>

4 Interface definition

The URL is a compiled string that specifies how PIE should be launched, and which module should be launched.

These are the main parts of the URL:

<protocol>://<hostname>/<webservice>?<parameters>

<protocol> http or https depending on the configuration of the web-server.
<hostname> The hostname of the Sectra PIE Service from where the Uploader, Viewer or Panel Module will be launched.
<webservice> The path to the web service of Sectra PIE,
<parameters> Several parameters can be passed on to the launched product to control how and what it should display. See URL parameter descriptions

4.1 URL parameter descriptions

This section lists the parameters of the URLs and their meaning. The parameters supplied must be encoded as specified in URL parameter encoding.

4.1.1 Parameter cmd
Parameter | Uploader | Viewer | Panel Module
---|---|---|---
`cmd` | X | X | X

Valid values are

- `PATHSEND` – Launches the Uploader
- `VIEWIMAGES` – Launches UniView Image Viewer
- `LAUNCHEXTERNAL` – Launches the Panel Module

### 4.1.2 Parameter `time`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Uploader</th>
<th>Viewer</th>
<th>Panel Module</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>time</code></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Timestamp for the creation of the URL. The timestamp is the system clock time measured in seconds since January 1st 1970 (see ANSI-C `time()` function).

**Note**: The timestamp should not be compensated for daylight saving.

### 4.1.3 Parameter `user_id`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Uploader</th>
<th>Viewer</th>
<th>Panel Module</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>User_id</code></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

The user name to be used to login to the launched product.

If the “`user_id`” parameter is omitted, the launched product will start with requesting the user to log on with user name and password before it launches the appropriate module.

### 4.1.4 Parameter SiteID

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Uploader</th>
<th>Viewer</th>
<th>Panel Module</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SiteID</code></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

The Site ID / Code.

### 4.1.5 Parameter CaseKey

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Uploader</th>
<th>Viewer</th>
<th>Panel Module</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CaseKey</code></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

The value of the CaseKey parameter should, when used, meet the format of `SiteNo-TNumber-HashKey`.

### 4.1.6 Parameter ReportID

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Uploader</th>
<th>Viewer</th>
<th>Panel Module</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ReportID</code></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

The value of the ReportID parameter should meet the format `SiteNo-TNumber`
4.1.7  Parameter ext_patinfo_url

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Uploader</th>
<th>Viewer</th>
<th>Panel Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext_patinfo_url</td>
<td>X</td>
<td>X</td>
<td>?</td>
</tr>
</tbody>
</table>

The ext_patinfo is the url for the Data Repository. This is required for the Viewer to retrieve the Patient details for the currently selected Slides. As this is a URL, which will be embedded within the main URL it will need to be URL Encoded.

4.1.8  Parameter ext_patinfo_acctoken

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Uploader</th>
<th>Viewer</th>
<th>Panel Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext_patinfo_acctoken</td>
<td>X</td>
<td>X</td>
<td>?</td>
</tr>
</tbody>
</table>

The access token that will be used in conjunction with ext_patinfo_url to ensure secure access to local patient data.

4.2  URL parameter encoding

The parameters for the IEP API are listed in the below table. The parameter values of the URL are only allowed to contain characters listed in Table 4.30, “Allowed characters in parameter values”. All other characters must be substituted with an escape sequence.

Table 1 Allowed characters in parameter values

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a-z</td>
<td>Lower case characters Upper case characters</td>
</tr>
<tr>
<td>0-9</td>
<td>Digits</td>
</tr>
<tr>
<td>.</td>
<td>Dot</td>
</tr>
<tr>
<td>-</td>
<td>Dash</td>
</tr>
<tr>
<td>_</td>
<td>Underscore</td>
</tr>
<tr>
<td>!</td>
<td>Exclamation mark</td>
</tr>
<tr>
<td>(</td>
<td>Left-hand parenthesis</td>
</tr>
<tr>
<td>)</td>
<td>Right-hand parenthesis</td>
</tr>
</tbody>
</table>

The escape sequences are character triplets, consisting of the percent character “%” followed by two hexadecimal digits representing the ASCII code of the substituted character. For example, the back-slash character “\” is substituted with “\%5C”.

**Note**: Character “^” should not be substituted when it is used as a list separator. substituted with an escape sequence.

5  How to compile the URL for EPR Integration(URL)

The following topics are included in this chapter:
- Access control key creation
- Compiling and launching the URL

The URL must be generated (compiled) and launched by an external application. This is done by implementing support for URL-generation into the code of the external application.
Please note that Encryption of the query string using PKi Encryption as described below is the preferred option as it facilitates Single Sign on, however the Access control control key is included for completeness.

5.1 Access control key creation

UniView and IEP normally include their own user access control functions, requiring a user to log on using a user name and password before access is granted. If however, the calling application includes its own reliable access control, the standard access control of the launched product can be bypassed. In that case the access control is made using a temporary access control key, supplied as a URL parameter.

The temporary access control key is calculated on data identifying the requested examination combined with a timestamp and a system password (see section 3.2 About the system password). The access control key is generated by computing the SHA1 hash of the concatenated URL parameter values, a time stamp and the system password. To generate the access control key use the following algorithm:

1. Create a concatenated string of all parameters. Escape sequence substitution should be applied to all parameters except the password parameter (system_password). Note that the order of parameters are of importance as the access control key must be generated with the same order of the parameters as they are passed in the URL. If these orders differs, the access control key will be erroneous. Also note that the password parameter (system_password) must be the last parameter in the concatenated string.
2. UTF-8 encode the concatenated string.
3. Compute the SHA1 hash on the UTF-8 encoded string.
4. Convert the SHA1 hash to a hexadecimal lowercase string.

**Note:** If multiple accession numbers or multiple examination ids that are separated with the `^` character are used, the `^` character itself should not be escaped when calculating the access control key, i.e. if "acc_no=acc/no1^accno2" the escaped string would be calulated as EscapeString("acc/no1") + "^" + EscapeString("accno2") = "acc%2fno1^accno2".

Below is a code snippet that shows a generic implementation (please note the parameters shown are not the actual ones used for PIE) of how to calculate the access control key:

```csharp
string concatenatedParams = "";
    // Note that the values shall be appended in the order they were passed in the URL
    // and not necessarily in the order shown here. See above for more information.
    // Escape sequence substitution should be applied to all parameters except the
    // password. (Not using all available parameters in the example code)
    concatenatedParams += EscapeString(stop);
    concatenatedParams += EscapeString(cmd);
    concatenatedParams += EscapeString(time);
    concatenatedParams += EscapeString(user_id);
    concatenatedParams += EscapeString(SiteID);
    concatenatedParams += EscapeString(CaseKey);
    concatenatedParams += EscapeString(ReportID);
    concatenatedParams += EscapeString(ext_patinfo_url);
    concatenatedParams += EscapeString(ext_patinfo_acctoken);
   concatenatedParams += system_password; // Note: No escape sequence for password string
    // UTF-8 encode the concatenated string
    byte[] utf8Bytes = GetUtf8Bytes(concatenatedParams);
    // Compute the SHA1 hash on the UTF-8 encoded string
    byte[] sha1Hash = ComputeSHA1Hash(utf8Bytes);
    // Convert byte array to hexadecimal lowercase string
    string accessKey = ByteArrayToHexString(sha1Hash);
```
The implementation of “EscapeString”, “GetUtf8Bytes”, “ComputeSHA1Hash” and “ByteArrayToHex” will vary depending on programming language.

**Note:** If the close_popup parameter is used, it should not be included in the concatenated string.

### 5.2 Compiling and launching the URL

With the access control key generated, what’s left is to compile and launch the URL. This section contains a full implementation of the EPR Integration (URL) flow, from access control generation to URL compilation and launching using both POST and GET requests, written in C#. For other examples of access control key generation, see appendix A Algorithm descriptions in an alternative programming language. Regarding launch of the URL, this is the only sample offered.

#### 5.2.1 About the example implementation

The code contains two helper classes, UrlLaunchString and WebLauncher, and a simple program that sets up a UrlLaunchString and then launches it through a WebLauncher.

The best way to examine the code is to look at it in an IDE. Following is an example how to do it.

**Building the code in Microsoft Visual Studio.**

1. Create a new project and choose Console Application.
3. Replace the code in Program.cs with the code in section 5.2.2 Example implementation and then build.
4. Add a break point in the beginning of the program's Main() method and start debugging by hitting F5.

```csharp
using System;
using System.Text;
using System.Web;
using System.Collections.Generic;
using System.Windows.Forms;
namespace UrlIntegration {
    class Program {
        [STAThread]
        static void Main(string[] args) {
            UrlLaunchString urlLaunchString = new UrlLaunchString() {
                BaseURL = "http://localhost:35555/External/apiLaunch?",
                UseHttpPost = true,
                Command = "PATHSEND",
                UserId = "myLisUser",
                CaseKey = "SiteNo-TNumber-Hash",
                SiteId = "34",
                ExtPatinfoUrl = "https://localhost:5443",
                ExtPatinfoAcctoken = "Hx24Yx",
                Stop = false,
                SystemPassword = "Test123",
            };
            WebLauncher launcher = new WebLauncher(urlLaunchString);
            launcher.Launch();
        }
    }
}
```
public WebLauncher(UrlLaunchString launchString) {
    LaunchString = launchString;
}
public void Launch() {
    if (LaunchString.UseHttpPost)
        LaunchHttpPost();
    else
        LaunchHttpGet();
}

/// <summary>
/// Launches the content of the URL launch string using http GET
/// </summary>
private void LaunchHttpGet() {
    string url =
        LaunchString.GetStartUrl() + "?" + LaunchString.GetQueryString();
    System.Diagnostics.Process.Start(url);
}

/// <summary>
/// Launches the content of the URL launch string using http POST
/// </summary>
private void LaunchHttpPost() {
    // Create web browser control
    webBrowser = new WebBrowser();
    webBrowser.Visible = false;
    // Create form to host the web browser
    webBrowserForm = new Form();
    webBrowserForm.Size = new System.Drawing.Size(0, 0);
    webBrowserForm.FormBorderStyle = FormBorderStyle.None;
    webBrowserForm.Controls.Add(webBrowser);
    webBrowserForm.Visible = false;
    // Prepare post data
    byte[] postData =
        Encoding.UTF8.GetBytes(LaunchString.GetQueryString());
    string additionalHeaders =
        "Content-Type: application/x-www-form-urlencoded"
        + Environment.NewLine;
    // Set timer to close form in 3 seconds. That should be
    // enough to send the request to the default browser.
    closeTimer.Tick += closeTimer_Tick;
    closeTimer.Interval = 3000;
    closeTimer.Start();
    // Navigate to the url and post the data. The "_blank"
    // frame target will force opening the URL in the
    // standard browser of the workstation.
    webBrowser.Navigate(new Uri(LaunchString.GetStartUrl()),
        "_blank", postData, additionalHeaders);
    Application.Run(webBrowserForm);
}

/// <summary>
/// When the close timer ticks, close all open forms.
/// </summary>
private void closeTimer_Tick(object sender, EventArgs e) {
    webBrowserForm.Close();
}

internal class UrlLaunchString {
    /// <summary>
    /// These fields govern the URL format
    /// </summary>
    public string BaseURL { get; set; }
    public bool UseHttpPost { get; set; }

    /// <summary>
    /// These fields govern the access control key
    /// </summary>
    public bool Stop { get; set; }
}
public string UserId { get; set; }
public string CaseKey { get; set; }
public string ReportID { get; set; }
public string SiteId { get; set; }
public string ExtPatinfoUrl { get; set; }
public string ExtPatinfoAcctoken { get; set; }
public string MrnIntegrationId { get; set; }
public string AccessNumberIntegrationId { get; set; }
public string Command { get; set; }
public string ExternalRequestID { get; set; }
public DateTime? DateAndTime { get; set; }
public string SystemPassword { get; set; }

public UrlLaunchString() { }

/// <summary>
/// Compiles the URL fields into a start URL
/// </summary>
public string GetStartUrl() {
    StringBuilder url = new StringBuilder();
    url.Append(BaseURL);
    return url.ToString();
}

/// <summary>
/// Adds a string to the parameters to be added
/// </summary>
public void AddStringParam(string input, string keyName, ref List<string> hashParams, ref List<string> retval) {
    if (String.IsNullOrEmpty(input)) return;
    var escapeStr = EscapeString(input);
    hashParams.Add(escapeStr);
    retval.Add(keyName + "=" + escapeStr);
}

/// <summary>
/// Adds a list to the parameters to be added
/// </summary>
public void AddListParam(List<string> input, string keyName, ref List<string> hashParams, ref List<string> retval) {
    if (input == null || input.Count <= 0) return;
    var parameterString = ListToString(input, '^', true);
    AddStringParam(parameterString, keyName, ref hashParams, ref retval);
}

/// <summary>
/// Compiles the UrlLaunchString into a query string
/// Note that the hashsum shall be calculated in the order they are passed in the URL
/// (Not using all available parameters in the example code)
/// </summary>
public string GetQueryString() {
    List<string> retval = new List<string>();
    List<string> hashParams = new List<string>();
    if (Stop) {
        retval.Add("stop=1");
        hashParams.Add("1");
    }
    AddStringParam(Command, "cmd", ref hashParams, ref retval);
    if (!string.IsNullOrEmpty(UserId)) {
        TimeSpan t = (DateTime.UtcNow - new DateTime(1970, 1, 1));
        long time = (long)t.TotalSeconds;
    }
}
var timeString = time.ToString();
    AddStringParam(timeString, "time", ref hashParams, ref retval);
}

AddStringParam(UserId, "user_id", ref hashParams, ref retval);
AddStringParam(SiteId, "SiteID", ref hashParams, ref retval);
AddStringParam(CaseKey, "CaseKey", ref hashParams, ref retval);
AddStringParam(ReportID, "ReportID", ref hashParams, ref retval);
AddStringParam(ExtPatinfoUrl, "ext_patinfo_url", ref hashParams, ref retval);
AddStringParam(ExtPatinfoAcctoken, "ext_patinfo_acctoken", ref hashParams, ref retval);
// System password must be added last to the parameters
// since it will be added last by the part verifying the
// launch string.
// System password will required if PKI encryption is not used
if (!String.IsNullOrEmpty(SystemPassword))
{
    hashParams.Add(SystemPassword);
    retval.Add("key=" + GetAccessControlKey(hashParams));
}
return ListToString(retval, '&', false);

/// <summary>
/// Strings might contain characters that are not supported in
/// a URL, so these need escaping.
/// </summary>
private string EscapeString(string str)
{
    return HttpUtility.UrlEncode(str);
}

/// <summary>
/// Compile a string with list items separated by a separator
/// character.
/// </summary>
private string ListToString(List<string> strList, char separator, bool escape)
{
    StringBuilder retval = new StringBuilder();
    foreach (string str in strList)
    {
        if (retval.Length > 0)
        {
            retval.Append(separator);
        }
        if (escape)
        {
            retval.Append(EscapeString(str));
        }
        else
        {
            retval.Append(str);
        }
    }
    return retval.ToString();
}

/// <summary>
/// Generate the access control key from a list of parameters.
/// </summary>
private string GetAccessControlKey(IEnumerable<string> hashParameters)
{
    if (hashParameters == null)
    {
        return String.Empty;
    }
    StringBuilder concatenatedParams = new StringBuilder();
    foreach (string param in hashParameters)
    {
        concatenatedParams.Append(param);
    }
    // The three steps from string of params to hexadecimal
/// string of SHA1 hash bytes.
byte[] utf8Bytes = GetUtf8Bytes(concatenatedParams.ToString());
byte[] sha1Hash = ComputeSHA1Hash(utf8Bytes);
string accessControlKey = ByteArrayToHexString(sha1Hash);
return accessControlKey;
} /// <summary>
/// Returns a byte array representing each character in
/// the input string as a UTF-8 encoded byte.
/// </summary>
private static byte[] GetUtf8Bytes(string parameters)
{
return Encoding.UTF8.GetBytes(parameters);
} /// <summary>
/// Computes a hashed byte array of the input using the
/// SHA-1 hash algorithm.
/// </summary>
private static byte[] ComputeSHA1Hash(byte[] utf8bytes)
{
SHA1Managed hasher = new SHA1Managed();
return hasher.ComputeHash(utf8bytes);
} /// <summary>
/// Converts a byte array into a hexadecimal lowercase
/// string of the concatenated bytes.
/// </summary>
private static string ByteArrayToHexString(byte[] sha1Hash)
{
StringBuilder hexString = new StringBuilder();
foreach (byte hashByte in sha1Hash)
{
    hexString.Append(hashByte.ToString("x2"));
}
return hexString.ToString();
}
}

For a complete Visual Studio project please contact Sectra.

Example URLs
( Note PKI Encryption of the URL is the preferred option )

Upload Pathology Images

https://uat.pacsportal.co.uk/IEPV65/External/apiLaunch?cmd=PATHSEND&time=1490346397&user_id=my_lis_user_id&SiteID=123&CaseKey=SITENo-TNo-Hash&ReportID=123&ext_patinfo_url=https%3a%2f%2flocalhost%3a5443&ext_patinfo_acctoken=Hx24Yx&key=616e9ce6e20609eb00e167f9f054e553f87a93c1

View Pathology Image(s) for a case

https://uat.pacsportal.co.uk/IEPV65/External/apiLaunch?cmd=VIEWIMAGES&time=1490346407&user_id=my_lis_user_id&SiteID=123&CaseKey=SITENo-TNo-Hash&ReportID=123&ext_patinfo_url=https%3a%2f%2flocalhost%3a5443&ext_patinfo_acctoken=Hx24Yx&key=7f33511276c757e1c6feb811e4f7cc67da54def1

Launch the Panel Module

https://uat.pacsportal.co.uk/IEPV65/External/apiLaunch?cmd=LAUNCHEXTERNAL&time=1490346416&user_id=my_lis_user_id&SiteID=123&CaseKey=SITENo-TNo-
Hash&ReportID=123&ext_patinfo_url=https%3a%2f%2flocalhost%3a5443&ext_patinfo_acctoken=Hx24Yx&key=82cca745d0ab36e17c4354a3c1283624a6226bd

Example PKI Encrypted URL

https://uat.pacsportal.co.uk/IEPV65/External/apiLaunch?SEUS=F1H15QODIaBahnN0e7HQk9IyWJN5VDfM7uSW6yWeQKIip7F1aiqM%2Fta8zDdUvYy0kkTllr%2fBKKW00v24pooy9jGidBbGa5iSHUMUsmq AwgbCu9GKJarWsBHJAOlBr1iDR5vhwWGSFH9sIn0BZk6b%2B5uPRgisDFb9kZSpshw9C0ov30HUxtbG3ui46b0aljTjkwCWWofzD69Rar%2fBYp5ofWtsO7gQwPTsydfWF9OQxZxMUIIUSAUGuxX1rqcte5nMkJT87mnC8mMCztp8bpcOIvtio3xAGMUbEPE5cI3t5ehw6S81HNBDhuHuhZDbNvVYw%2FeZYg%2fBCIxzSj449a5BoyCuw4fDPVRVOiuYCecG3jczQxV8JGB8436fhr%2Bat5khUaYP8jE%2f2fIQwl62EaJnNceqso1EQ5S%2fFyHthcufoacDCqhV8eqqS4RtmnOo7PNpkjOFZ%2fFINQiUKCHVLCuVIIe7lpjA1BPkosSOvsKwZpDD%2fFGK6W%2f11ws1lTrCtvH4atxMeMZOE222Gi4OVSz%2fBc9s91YbyFyPNC2wbv6smIbCoCFWCDFLT9efp4WyOObEHb&ESP=pSRV83Uy8yXIIHPZhp0R2CYGQmpaVU0dsTL9jvb4e4c%2B3PEdEmehGOKmHUOPVTAGV%2fBBB7lmmp1mV%2fFC%2fBH2gJMr%2FBwrsLiSx%2fBWBvYMQoUXVzSgLWlp4btfzVSGl%2fBFID%2fF25EVpmX%2fFxrIz%2fFpy%2fFYQ7tpKGyjA7eAvb%2fFRSIpajT%2fBqNVSRnsEWGEH3CRFJN03%2fFZxmdJLz7zUDPsoY0naOGD0dWlZ2Y9yvo1zdYl3yO8Gdaeu1bKDs0v4G0eh20jEt%2fB%2fFYyp52xvjdl96Yb5aeExKxFileif2uT7sjuBStzX7mWQj8wDVY5ulb%2fBxKzp2Sw%2fBNH7M%2fBUAIQ%2fB9uQIA%3D%3D&IV=AuFMH%2fFyLICBVvOqKRza0pg%3D%3D&DV=1

6 PKI encoding

The following topics are included in this chapter:
- About EPR Integration (URL) using PKI
- Getting started with EPR Integration (URL) using PKI
- About the provided software for securing URLs
- About Certificate handling
- Fout! Verwijzingsbron niet gevonden.

6.1 About EPR Integration (URL) using PKI

EPR Integration (URL) using PKI is a more secure version of EPR Integration (URL), where the PKI technology is used. Instead of transmitting a hashed master password in the URLs, public and private keys are used to sign, encrypt, decrypt and verify the content and sender of the different URLs. This additional security layer is wrapped around the existing functionality.

Please note this is the preferred method of use as it allows Single Sign On where as the hash key approach will require the user to login details.

With only a few exceptions, the URLs used in EPR Integration (URL) remain the same as in ordinary EPR Integration (URL) before they are encrypted.

Differences in the URL before encryption for EPR Integration (URL) using PKI

- The key parameter is not needed
- Whether to log in automatically or not is determined by the presence of the user_id and time parameters

An automatic login will be performed if, and only if, both user_id and time are supplied in the URL.
Renewing an already existing session also requires the time parameter to be present in the URL.

- The close_popup parameter may not be part of the encrypted part of the URL.

It is still possible to use it, but it must remain unencrypted and be appended to the URL after the encrypted part.

EPR Integration (URL) using PKI was introduced in 19.1 and is applicable for IDS7 and UniView.

### 6.2 Getting started with EPR Integration (URL) using PKI

First follow the instructions found chapter 2 Getting started, and then perform the required actions in the following steps.

#### 6.2.1 Create and install the certificates

First the required certificates must be created and installed. The following certificates are needed.

- Certificates on the Sectra IEP Server host
  - A certificate belonging to the Sectra IEP Server, containing a public and private key.
  - Certificates belonging to all different integrating parties, where each contains that integrating party's public key.
- Certificates on the server host of each integrating party
  - A certificate belonging to the Sectra IEP Server, containing a public.
  - The integrating party's own certificate, containing a public and private key.

Make sure the all these certificates adhere to the following requirements:

Each certificate should be...
- Created using an encryption key of length 2048 bits.
- Created using SHA-256 or SHA-512 hash function.

Installing the certificates

1. Place the certificate you wish to install on the host you wish to install it on
2. Select the certificate in the File Explorer and press enter.
3. Select the store location of the certificate
4. Press Next
5. If the certificate contains a private key that is protected by a password, enter that password.
6. Press Next
7. Select Place all certificates in the following store
   Select Browse and select a store
    Press Next
8. Press Finish

**Note:** Be aware of what users will need to access the certificates.

Certificates installed in a store in the current user store location may only be accessible by the user that installed the certificate. One way of making them accessible to all users on the host is to install the certificates in the local machine store location. Doing so requires a user with administrative rights.
Managing the access rights of a certificate private key installed in the local machine store location

Private keys belonging to certificates installed in a certificate store in the local machine store location are not readable by default. It is necessary to give read permission to the user that needs to access those private keys. For instance on the Sectra IEP Server host the user running the IEP Application, will need to access the private key of the certificate belonging to the Sectra IEP Server.

1. Log on to the host where the certificate is installed as a user with administrative rights.
2. Start the mmc application
   Press the Windows button and write "mmc"
   Press Enter
   If a prompt regarding user access control appears, accept it by pressing Yes
3. Add the Certificates snap-in to the mmc application
   Select File->Add/Remove Snap-in...
   Select the Certificates alternative in the list
   Press OK
   Select the Computer account certificate store location
   Press Next and then Finish
   4. Open the certificate's private key access control window
      Browse to the certificate store where the certificate is installed in the menu on the left
      Right click on the certificate to modify and select All Tasks->Manage Private Keys...
      5. Give the desired user permission to read the certificates private key
         Press Add
         Enter the name of the user in the name field
         Press OK

6.3 About the provided software for securing URLs

There are two different ways for securing URLs, by using the provided commandline program or by developing your own software using the Sectra SecureURL NuGet Package. Both alternatives provide the user with two functions that can be called.

• Secure - This function will sign and encrypt an input string. It takes information about where to find the necessary certificates as input. Typically used by the party that wants to URL launch one of Sectra's products.

• View - This function will decrypt and verify the author of an input string. It takes information about where to find the necessary certificates as input. Typically used by the Sectra Healthcare Server in order to get the information in the URL and verify the sender.

6.3.1 Common requirements

Both the commandline program and the NuGet package will need to access certificates containing the necessary keys. These certificates will therefore need to be installed in the system in order to create a secure URL.
See also
• section Create and install the certificates

6.3.2 SectraSecureURL commandline program

This section describes the functions, input and output of the SectraSecureURL commandline program and gives some example usages.
Table 6.1 Sectra.SecureUrl Commandline program overview

<table>
<thead>
<tr>
<th>Function</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Secure</strong></td>
<td>-t, --sectra-cert-store (Default: AddressBook) Specification of what store to get the Sectra certificate from. -o, --integrating-cert-store (Default: MY) Specification of what store to get the integrating party certificate from. -c, --cert-identifier (Default: FindBySubjectName) Specification of what property of the certificate to use to identify the certificates, accepts values such as FindBySubjectName and FindByThumbprint. -e, --sectra-cert-identification (Default: SectraSecureUrlLaunch) Identification of the Sectra certificate. Value depends on the CertificateIdentificationType. -i, --integrating-part-cert-identification Required. Identification of the integrating part's certificate. Value depends on the cert-identifier. Value pos. 0, URL launch access string to use.</td>
<td>A signed and encrypted URL access string.</td>
</tr>
<tr>
<td><strong>View</strong></td>
<td>-t, --sectra-cert-store (Default: AddressBook) Specification of what store to get the Sectra certificate from. -o, --integrating-cert-store (Default: MY) Specification of what store to get the integrating party certificate from. -c, --cert-identifier (Default: FindBySubjectName) Specification of what property of the certificate to use to identify the certificates, accepts values such as FindBySubjectName and FindByThumbprint. -e, --sectra-cert-identification (Default: SectraSecureUrlLaunch) Identification of the Sectra certificate. Value depends on the cert-identifier. -i, --integrating-part-cert-identification Required. Identification of the integrating part's certificate. Value depends on the cert-identifier. Value pos. 0, URL launch access string to use.</td>
<td>A decrypted and verified URL access string.</td>
</tr>
</tbody>
</table>
To encrypt a URL access string using the SectraSecureURL commandline program

1. Enter the directory of the commandline program

   C:\> cd SectraSecureUrl\bin

2. Run the program with the Secure command

   C:\SectraSecureUrl\bin> \Sectra.SecureUrl.exe secure -e 'Url Launch System' -i 'Integrating Url Launch Party' -t 'my' -o 'my' -c 'FindBySubjectName' 'myUrlAccessString'

To decrypt a URL access string using the SectraSecureURL commandline program

1. Enter the directory of the commandline program

   C:\> cd SectraSecureUrl\bin

2. Run the program with the View command

   C:\SectraSecureUrl\bin> \Sectra.SecureUrl.exe view -e 'Url Launch System' -i 'Integrating Url Launch Party' -t 'my' -o 'my' -c 'FindBySubjectName' 'SEUS=Vj2Ifx5%3D%3D&ESP=o%2BdU%3D&IV=Y%2FeJcmwrXHaVNP6drf62g%3D%3D'

See also
• section Fout! Verwijzingsbron niet gevonden.

6.3.3 SectraSecureURL NuGet Package

This section describes the functions, input and output of the SectraSecureURL NuGet package and gives some example usages.

Table 6.2 Sectra.SecureUrl NuGet package overview

<table>
<thead>
<tr>
<th>Function</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure</td>
<td>string urlLaunchString</td>
<td>EncryptedUrlString</td>
</tr>
<tr>
<td></td>
<td>RSACryptoServiceProvider</td>
<td>encryptedUrlString</td>
</tr>
<tr>
<td></td>
<td>integratingPartCert.PrivateKey</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PublicKey sectraCert.PublicKey</td>
<td></td>
</tr>
<tr>
<td>View</td>
<td>EncryptedUrlString string urlLaunchString</td>
<td>string decryptedUrlString</td>
</tr>
<tr>
<td></td>
<td>RSACryptoServiceProvider</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sectraCert.PrivateKey</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PublicKey integratingPartCert.PublicKey</td>
<td></td>
</tr>
</tbody>
</table>

To encrypt a URL access string using the SectraSecureURL NuGet package in C#

1. Make your own code to fetch the required certificates

   X509Certificate sectraCert = GetSectraCertificate();
   X509Certificate integCert = GetIntegratingCertificate();

2. Deserialize the encrypted URL string into an object of the type EncryptedUrlString using the SectraSecureURL NuGet package
EncryptedUrlString encryptedUrlString =
EncryptedUrlString.DeserializeUrlString(encryptedUrl);

3. Call the View function in the SectraSecureURL NuGet package

var securityNuget = new UrlLaunchSecurity();
string encryptedString =
securityNuget.View(encryptedUrl, (RSACryptoServiceProvider)integ.PrivateKey,
sectraCert.PublicKey);

6.4 About Certificate handling

EPR Integration (URL) using PKI replaces the the security feature of a shared master password with public and private keys. These public and private keys are to be stored in certificates on the hosts that require them, hence the need to handle certificates.

When utilizing EPR Integration (URL) in general many different client hosts may each try to start different applications. This is true for EPR Integration (URL) using PKI as well. It must be noted, however, that the means to generate the URLs must not be spread out to these different client hosts when using EPR Integration (URL) using PKI. Generating a valid URL for EPR Integration (URL) using PKI requires one private key and one public key, located in two different certificates. The private key will be the secret of the integrating party, and must stored with the highest confidentiality on some central server host. The clients will therefore have to request URLs from some service on this central node. While sharing the private key between all clients would allow them to generate the URLs on their own, this would be a major security flaw as it would greatly increase the risk of the private key being compromised. Anyone able to obtain this private key would be able to impersonate the integrating party and create valid URLs on their own, thus accessing the system in the integrating party's name.