Secure HLS streams with AES-128 external encryption using the Wowza Streaming Cloud REST API

The Wowza Streaming Cloud™ service allows you to secure HLS streams using the external method of AES-128 encryption. When you use the external method of AES-128 encryption, encryption keys are delivered to devices from an external URL. This article describes how to use the Wowza Streaming Cloud REST API to configure AES-128 encryption for an HLS stream.

Configure AES-128 encryption

1. Using the Wowza Streaming Cloud REST API, do one of the following:

   - Create a live stream. For instructions, see Get started broadcasting a live stream using the Wowza Streaming Cloud REST API.
   - Create a transcoder and add output renditions. For instructions, see the following articles:
     - Create an ABR stream and send it to a target with the Wowza Streaming Cloud REST API
     - Pass a stream through the transcoder to a target with the Wowza Streaming Cloud REST API

2. Configure the following transcoder properties to enable AES encryption:

<table>
<thead>
<tr>
<th>Section</th>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cupertino</td>
<td>aes128Host</td>
<td>string</td>
<td>Specifies the URL that devices will use to fetch the key to decrypt the stream.</td>
</tr>
<tr>
<td>Cupertino</td>
<td>aes128Key</td>
<td>string</td>
<td>Specifies the 16-byte key that will be</td>
</tr>
</tbody>
</table>
used to decrypt the stream. The key must be 32 characters in length and can only contain hex characters (a-z, A-Z, 0-9). The key must match the key returned by the `aes128Host`.

Configure the `aes128Host`:

```
curl -X POST \
-H "Content-Type: application/json" \
-H "wsc-api-key: ${WSC_API_KEY}" \
-H "wsc-access-key: ${WSC_ACCESS_KEY}" \
-d '
  "property": {
    "key": "aes128Host",
    "section": "cupertino",
    "value": "https://[server-url].com/aes128/key.bin"
  }
}' "${WSC_HOST}/api/${WSC_VERSION}/transcoders/[transcoder_id]/properties"
```

The request updates the `aes128Host` property for the transcoder. The details of the transcoder property are listed in the response, which should look something like this:

```
{
  "property": {
    "key": "aes128Host",
    "section": "cupertino",
    "value": "https://[server-url].com/aes128/key.bin"
  }
}
```

Configure the `aes128Secret`:
The request updates the \texttt{aes128Secret} property for the transcoder. The details of the transcoder property are listed in the response, which should look something like this:

```
{
    "property": {
        "key": "aes128Secret",
        "section": "cupertino",
        "value": "B1C014C12TFL031DEVL91345GI8211DD"
    }
}
```

\textbf{Note:} If you have started a transcoder at any point before updating its transcoder property, you must reset the transcoder for the property to take effect. This step isn’t necessary if you haven’t started the transcoder at all.

Reset a transcoder

```
curl -X PUT \
-H "wsc-api-key: ${WSC_API_KEY}" \
-H "wsc-access-key: ${WSC_ACCESS_KEY}" \
"${WSC_HOST}/api/${WSC_VERSION}/transcoders/[transcoder_id]/reset"
```

\textbf{Examples}

The following examples show how to use popular web application technologies such as \texttt{ASP.NET}, \texttt{JSP}, and \texttt{PHP} to send the key data. Each example includes a Boolean \texttt{isValid} value that defaults to \texttt{true}. You can modify the examples to provide...
your own security tests to validate that users can access the content. If users shouldn’t be allowed to access the content, you can block them from receiving the decryption key by setting `isValid` to `false`.

If the request for the key returns a status of `403`, then the device won’t be able to decrypt and play the stream. If the key is returned, then the stream will be decrypted and played. Require HTTPS access to this key to ensure that it isn’t sent over an unsecured connection on the Internet.

The key being sent in these examples is `DE51A7254739C0EDF1DCE13BBB308FF0`. You should substitute this value with a different 16-byte key. The key should match the key value specified for the `aes128Secret` transcoder property.

**Note:** These examples are provided as-is with no expressed warranty. You can modify or distribute them without restriction.

**ASP.NET example**

```csharp
<%@ Page Language="C#" %>
<%
    Boolean isValid = true;
    if (!isValid)
    {
        Response.Status = "403 Forbidden";
    }
    else
    {
        Response.AddHeader("Content-Type", "binary/octet-stream");
        Response.AddHeader("Pragma", "nocache");

        String keyStr = "DE51A7254739C0EDF1DCE13BBB308FF0";
        int len = keyStr.Length/2;
        byte[] keyBuffer = new byte[len];

        for (int i=0;1
```
<%@ page import="java.util.*,java.io.*" %>
<%

boolean isValid = true;
if (!isValid)
{
    response.setStatus(403);
}
else
{
    response.setHeader("Content-Type", "binary/octet-stream");
    response.setHeader("Pragma", "no-cache");

    String keyStr = "DE51A7254739C0EDF1DCE13BBB308FF0";

    int len = keyStr.length()/2;
    byte[] keyBuffer = new byte[len];

    for (int i=0; i

PHP example

More resources

- Test AES encryption for HLS streams from the Wowza Streaming Cloud REST API