# DTCP-IP For DLNA

## Table of Contents



## Change Log

| Name | Description | Date |
|---|---|---|
| Kevin Arruda | Created Initial Document | 01/03/2008 |

# DTCP-IP For DLNA

## What is DTCP-IP?

DTCP-IP (Digital Transmission Content Protection – Internet Protocol) is a Link Protection Technology that has been specifically adapted for IP (Internet Protocol) transport. For more information on Link Protection in general, please see the "Overview of Link Protection in DLNA" White Paper. DTCP alone was originally designed for IEEE 1394 (FireWire) and USB transport links, but was later mapped to TCP/IP for use over home network Ethernet/WiFi links. This is the implementation that the DLNA has utilized.

A formal definition of DTCP-IP is available here from the DTLA. It is recommended that you review this document before continuing with this White Paper. You should be familiar with what a "PCP" (Protected Content Packet) is, as well as what "AKE" (Authentication and Key Exchange) is, and why it is necessary.

## How is DTCP-IP used within DLNA?

DTCP-IP is integrated into the DLNA specification by the Link Protection Guidelines, which establish the additional protocol elements necessary to facilitate it's use. The more important modifications are as follows:

The MIME type of DTCP protected media resources is appended with new data to facilitate the AKE (Authentication and Key Exchange) process. This new MIME replaces the media's previous MIME type in all instances where it is transmitted (e.g. in an HTTP header or in the CDS metadata of a DMS). An example of a DLNA DTCP MIME type is shown below:

1.) Original MIME-type: 'audio/mp3'
DLNA DTCP MIME-type: 'application/x-dtcp1;DTCP1HOST=<host>;DTCP1PORT=<port>;CONTENTFORMAT=<MIME-type>"
Where:
<host> = I.P. Address that the Source Device is listening for AKE requests on.
<port> = The TCP Port that the Source Device is listening for AKE requests on.
<MIME-Type> = The original MIME type of the media, in this case 'audio/mp3'.
So, the MIME-type of this example media item could look something like the following:
'application/x-dtcp1; DTCP1HOST= 192.168.0.5;DTCP1PORT=8000;CONTENTFORMAT=audio/mp3'

2.) When performing a subset request for a video item (e.g. a Range or TimeSeekRange.dlna.org request, the Source must align the returned PCP (Protected Content Packet) to specific points, such as a GOP or VOB boundary in MPEG content, or a packet boundary in ASF content. This ensures that any request for a protected video item returns data that is possible to render without further requests.

3.) Since encryption and packetizing of protected content adds artificial length to media item on the network layer, new HTTP headers are defined to allow requests for a portion of the original media item. For DTCP-IP, the new HTTP headers are Range.dtcp.com (request) and Content-Range.dtcp.com (response). When supported by both devices, these headers allow requests for data from the original media item, which is smaller in size than the data that is transmitted over the network. A simple example is shown below:

Media item: myProtectedSong.mp3
Size : 3,000 KB.

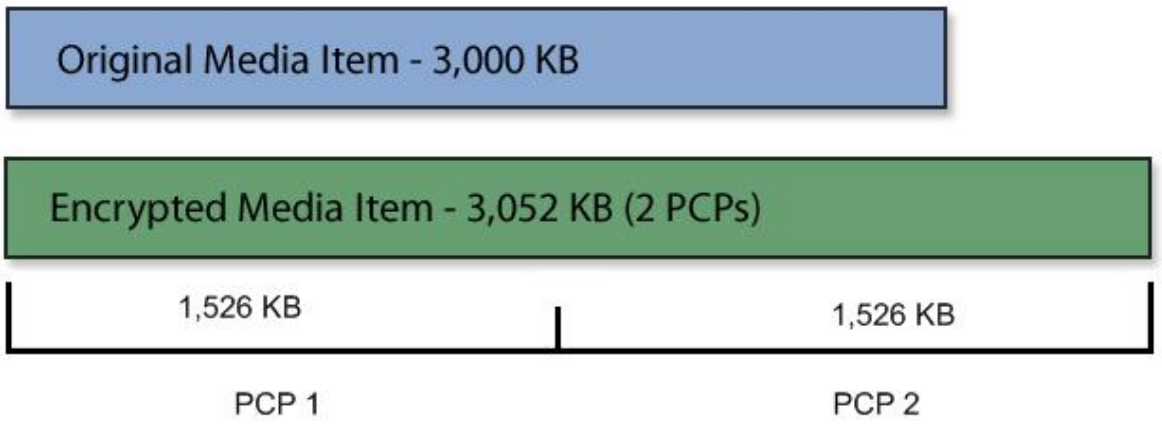


Fig 1: Example Content Length Change for Encrypted Media

As you can see from the diagram, there is a small but significant difference in the size of encrypted vs. non encrypted media. To refer to the different representations of the media, we commonly refer to two "byte domains" – the "Cleartext Byte Domain" and the "Network Byte Domain". The Cleartext Byte Domain refers to the bytes of the original media item – in this case [0-2999]. The Network Byte Domain refers to the encrypted form of the media item which is transmitted over the network – in this case [0-3051].

The additional bytes in the Network Byte Domain come from the PCP header, as well as padding bytes which are added to mathematically facilitate encryption. In the example above, we used two PCPs to encapsulate our content. We could have used any number, from 1 to 3,052 (if we put 1 byte in each PCP).

Now, let's say we are a rendering device and we would like to request a piece of the media item for rendering. We're not sure exactly how much data we want, but we know we need at least 1,000 bytes to start playing the song because that's how our rendering device works.

First, we'll try to request just 1,000 bytes of the item with a regular Range request.

Request:

GET /myProtectedSong.mp3
Range: bytes=0-999
TransferMode.dlna.org=Streaming

Response:
HTTP/1.1 206 Partial Content
Content-Range: 0-999/ 3125248
Content-Length:1000
TransferMode.dlna.org=Streaming

What was returned? Well, because we requested 1000 bytes from the network domain, we got 1000 bytes of our first PCP. Since we didn't use a Cleartext Request, the Source just gave us the network bytes we requested. When our rendering device tries to decrypt the PCP for rendering, it fails. What happened? Well we can see from the above diagram that the first PCP the DMS made was 1,526 KB in length, and we only requested 1,000 bytes of it. Without the entire PCP, we can't decrypt it. We could either read the PCP length out of the PCP header and request the rest of the bytes, or we could try a Cleartext Request. Let's try a Cleartext Request.

Request:
GET /myProtectedSong.mp3
Range.dtcp.com: bytes=0-999
TransferMode.dlna.org=Streaming

Response
HTTP/1.1 206 Partial Content
Content-Range.dtcp.com: bytes=0-999/ 3125248
Content-Length: 1022
TransferMode.dlna.org=Streaming

Now what was returned? Well, since we requested 1000 bytes from the original media item using the Cleartext Request Header, the DMS repackaged exactly 1,000 bytes from the media into one PCP and gave it back to us instead of using it's predetermined PCPs. Notice that our Content-Length is now 1022 for the 1000 bytes we requested. This is our 1000 bytes + 14 bytes for the PCP header + 8 padding bytes for the encryption. Now we decrypt that PCP successfully since we have a complete PCP, and we can begin rendering our content.

What have we done here? By using the Cleartext Request Header, we allowed our rendering device to operate more predictably, since if we know how much actual content data we are receiving, we can calculate with greater precision how much buffer space we need, how much decryption overhead there will be, as well as exactly how much media we can provide to the user and how long we have until we need the next chunk of media.

## What does this mean for a Developer?

Implementing DTCP-IP on a DLNA device provides a great deal of power and security to the device. Just like any DLNA certified device should interoperate with another DLNA certified device, a DTCP-IP DLNA device should interoperate with any other DTCP-IP DLNA device. While this may seem somewhat trivial, this assertion grants a good deal of power to the device. It means that the device can securely communicate and render protected content from **any** other DTCP-IP enabled DLNA device. It potentially grants the power to remove the traditional restrictions of Premium Content, which are usually limited to a small number of devices or mediums. For example, if all your media equipment were DTCP-ip DLNA certified, you could potentially buy licensed music online with your PC, and then render/transfer it to your MP3 player, home stereo, cell phone, PDA, television, automobile, etc. without fear of violating the media license or running into annoying playback restrictions. While this interoperability is not guaranteed (since it starts to blend with DRM (Digital Rights Management) – which is out of scope for DLNA), DLNA with DTCP-IP could certainly facilitate this scenario by securing the home network link between devices.