APPENDIX F

URL Signing and Validation

This appendix describes the URL signing and validation method for the Cisco Internet Streamer CDS. This appendix contains the following sections:

Introduction

The Cisco Internet Streamer CDS accepts and fulfills requests for video content from client devices in the form of content URLs. Content and service providers, in order to protect their copyright and fulfill their licensing obligations, often need to restrict access to content and limit viewing times. Basic authentication and authorization at the portal (for example, username and passwords) can help achieve this objective by restricting content access to authorized users. However, because URLs are inherently open, users (once authenticated at the portal) could potentially share these content URLs with other possibly unauthorized users, or continue to access the content beyond the allotted time.

Cisco Internet Streamer CDS provides the infrastructure to sign and validate content URLs, restricting access to some users and limiting viewing times.

URL Signing Components

One of the easiest ways to restrict content access to a particular user is to embed, within the content URL, the client IP address of the user for whom the content access was authorized. Similarly, to ensure that the content expires after a predetermined time, an expiry timestamp could be embedded. These values can then be validated against the actual client sending the request and the current time at the Service Engine serving the request. If either of the two validations fail, the request is rejected.

Note You can exclude the checks for the client IP address and the content expiry by configuring a service rule on each SE. For more information, see the "Configuring URL Signing" section on page 4-25.

However, because any of these strings in the URL could potentially be edited manually and circumvented by any knowledgeable user, it is important to generate and attach a signature to the URL. This can be achieved by attaching a keyed hash to the URL, using a secret key shared only between the signer (the portal) and the validating component (CDS).

CDS has incorporated an open and well-documented signing mechanism that uses standard hashing schemes. The URL signing mechanism offers the flexibility to either use the provided signing script, or you can develop a signing application in the platform or language of your choice, as long as it adheres to the specified format.

For signing and validation of the URL, the CDS relies on a set of one or more secret keys shared between the portal and the devices within the CDS.

Note Sometimes media players append the port number to the URL. In this case, the SE removes the port number from the URL before validating the signature.

Supported Protocols and Media

The URL signing and validation is supported across all CDS protocol engines: Windows Media Engine, Movie Streamer Engine, Flash Media Streaming Engine, and Web Engine.

Configuring the CDS for URL Signing

To enable validation of URLs in the CDS, the following tasks must be completed on all participating Service Engines:

- Configure shared secret keys
- Configure pattern-lists to match URLs, domain names, or both
- Configure rules to validate URLs matching the above pattern-lists
- Enable rules processing

Details on these configurations are available in the "Configuring URL Signing" section on page 4-25 and the "Configuring Service Rules" section on page 4-15.

The CDS URL signing infrastructure supports multiple keys. Different pieces of content, with different URLs, can be signed by different keys. Keys are stored as a key matrix and identified (indexed) by a key ID owner and a key ID number.

URL Signing Script

At the portal, URLs can be signed for a particular user (client IP address) and expiry time using a URL signing script. The URL signing script example included in this section requires Python 2.3.4 or higher.

URL Signing Version

The URL signing script offers three different versions:

- MD5 hash algorithm
- SHA-1 hash algorithm
- SHA-1 hash algorithm with the protocol removed from the beginning of the URL

When a URL is signed for RTSP and a player does a fallback to HTTP for the same URL, the validation fails because the URL signature includes RTSP. If the URL signature does not include the protocol, the fallback URL is validated correctly even though the protocol is HTTP.

If you do not specify a version for the script, MD5 is used and the SIGV string in the script is not added.

Following is an example of the URL signing script using the MD5 security hash algorithm:

python cds-ims-urlsign.py http://www.cisco.com/index.html 8.1.0.4 200000 1 2 cisco

An example of the resulting signed URL follows:

```
http://www.cisco.com/index.html?IS=0&ET=1241194518&CIP=8.1.0.4&KO=1&KN=2&US=deebacde45bf71
6071c8b2fecaa755b9
```

If you specify version 1 for the script, SHA-1 is used and the SIGV=1 string is added.

Following is an example of the URL signing script using the SHA-1 security hash algorithm:

python cds-ims-urlsign.py http://www.cisco.com/index.html 8.1.0.4 200000 1 2 cisco 1

An example of the resulting signed URL follows:

```
http://www.cisco.com/index.html?SIGV=1&IS=0&ET=1241194679&CIP=8.1.0.4&KO=1&KN=2&US=8349348
ffac7987d11203122a98e7e64e410fa18
```

If you specify version 2 for the script, SHA-1 is used. The protocol from the beginning of the URL is also removed before the signature is generated, and the SIGV=2 string is added. The protocol is RTSP, HTTP, or RTMP. The URL is signed without the protocol, but the final signed URL is printed with the protocol.

Following is an example of the URL signing script using the SHA-1 security hash algorithm with version 2 specified:

python cds-ims-urlsign.py http://www.cisco.com/index.html 8.1.0.4 200000 1 2 cisco 2

An example of the resulting signed URL follows:

```
http://www.cisco.com/index.html?SIGV=2&IS=0&ET=1241194783&CIP=8.1.0.4&KO=1&KN=2&US=68b5f5e
d97d1255a0ec42a42a4f779e794df679c
```

Example of a Python URL Signing Script

The following simple Python script demonstrates how to construct and sign URLs for use with the Internet Streamer CDS. This example script produces signatures compliant with the format used by the Internet Streamer CDS.

Depending on where the Python binary is installed, you may need to modify the first line of the script. The first line is only necessary if you plan to run the script as an executable. However, if you run the script using the Python interpreter, as documented in the "Running a Python URL Signing Script" section on page F-6, the first line is not required.

Note Sometimes media players append the port number to the URL. The port number is always removed when generating the URL signature.

```
#!/usr/local/bin/python
import md5
import hmac
import sha
import socket
import time
import sys
import optparse

def remove_port(url):
        """ Removes the port number from URL and then constructs the
        URL without port. """
        sep=":"
        arr=url.split(sep)
        url_no_port=arr[0]+sep
        if ( (len(arr)) == 3 ):
                url_no_port=url_no_port+arr[1]+"/"
                rem=arr[2].split("/",1)
                url_no_port=url_no_port+rem[1]
                return url_no_port
        else:
                return url

def sign_url(url,key):
        """ Signs url using key and returns the signed URL with the
        signature appended. """
        # Generate a MD5 hash of the key string (not the url)
        foo = md5.new(key)

        # Update the hash generated with the url string
        # This effectively means concatenating key and url and generating
        # a hash for the two
        foo.update(url)

        print "MD5 is used"
        # Get the digest in hex format (human readable)
        return url+foo.hexdigest()

def sign_url_sha1(url,key):
        """ Signs url using key and returns the signed URL with the
        signature appended. """
        foo = hmac.new(key,url,sha)
        print "SHA1 is used"
        return url+foo.hexdigest()

def usage():
        """ Prints usage for the URL signing script. """
        print "Usage:"
        print "python cds-ims-urlsign.py <url> <client-ip> <expiry-delay-seconds> 
<key-id-owner> <key-id-number> <key> <version>"
        print "Example:"
        print "python cds-ims-urlsign.py rtsp://abc.com/content/Apocalypto.mov 
171.71.50.123 120 1 2 BubbaGump \"\"or1or2"

if __name__ == "__main__":
        """ Prints signed URL """
        parser = optparse.OptionParser()
        (options, args) = parser.parse_args()
        if len(sys.argv) < 7:
                usage()
                sys.exit(2)
        url = sys.argv[1]               # URL
        client_ip = sys.argv[2]
        delay_seconds = sys.argv[3]     # Number of seconds after which URL expires
        ko = sys.argv[4]                # Key ID Owner
        kn = sys.argv[5]                # Key ID Number
        key = sys.argv[6]               # Key
        if ( len(args) == 6 ):
                version = ""
        else:
                version = sys.argv[7]
                version=int(version)

        # Set expiry time as current time (seconds since epoch) + delay
        et = time.time() + int(delay_seconds)
        expires = str(int(et))

        # Based on version we need to generate URL syntax for signing
        # By default v="" , which means sign URL with schema and no SIGV added. Uses md5
        # If v=1 , which means sign URL with schema and SIGV=1 added. Uses sha1
        # If v=2 , which means sign URL without schema and SIGV=2 added. Uses sha1
        if ((version!="") & (version!=0) & (version!=1)):
                # python-2.5 is needed for partition. So will use split
                #schema,sep,url = url.partition(':')
                sep="://"
                arr=url.split(sep)
                url = sep + arr[1]
                print url

        if url.find('?')==-1:
                url2 = url + "?"
        else:
                url2 = url + "&"

        url2 = remove_port(url2)

        # This string format is fixed and should not be modified.
        # Note that we sign even the "&US=" part that will point to the signature
        # If schema=1 then we need to append ver string as 0.
        if (version==""):                url2 = url2 + 
"IS=0"+"&ET="+expires+"&CIP="+client_ip+"&KO="+ko+"&KN="+kn+"&US=";
        elif (version==0):
                url2 = url2 + 
"IS=0"+"&ET="+expires+"&CIP="+client_ip+"&KO="+ko+"&KN="+kn+"&US=";
        elif (version==1):
                url2 = url2 + 
"SIGV=1"+"&IS=0"+"&ET="+expires+"&CIP="+client_ip+"&KO="+ko+"&KN="+kn+"&US=";
        elif (version==2):
                url2 = url2 + 
"SIGV=2"+"&IS=0"+"&ET="+expires+"&CIP="+client_ip+"&KO="+ko+"&KN="+kn+"&US=";
```

```
# Based on the Version decide the sign Method to call
if ((version=="") | (version==0)):
        url3 = sign_url(url2,key)

if ((version ==1) | (version==2)):
        url3 = sign_url_sha1(url2,key)

#After we sign, if version=2 we add schema to signed URL
if version==2:
        url3=arr[0]+url3
print url3
```

Running a Python URL Signing Script

The example script can be used as follows:

python cds-ims-urlsign.py <*url*> <*client-ip*> <*expiry-delay-seconds*> <*key-id-owner*> <*key-id-number*> <*key*> <*version*>

Syntax	Description
url	URL to sign.
client-ip	IP address of the client for which this URL is being signed, in dotted decimal format (A.B.C.D). The signed URL will be rejected if sent from any other client when signature validation is enabled.
expiry-delay-seconds	Seconds (from now) when the URL expires. The request will be rejected if the time period has passed when the URL is validated at the device. See the "Importance of Device Synchronization" section on page F-7.
key-id-owner	The first index into the key matrix. Valid entries are from 1 to 32.
key-id-number	The second index into the key matrix. Valid entries are from 1 to 16.
key	Shared secret key corresponding to this ordered pair (*key-id-owner*, *key-id-number*).
version	Hash algorithm used to generate the URL signature. The version number for the hash algorithm is as follows: • 0 or none—To use MD5, enter zero (0) or do not enter a version number. • 1—To use SHA-1. • 2—To use SHA-1 and remove the protocol from the URL before signing.

In addition to the above six variables, the current time is used to generate the URL signing, so even if the same values were used for the above variables, the signed URL would be different.

Note The client IP address and the content expiry are checked during validation based on the configuration of the service rules on the SE. For more information, see the "Configuring URL Signing" section on page 4-25.

To use the URL signing script on the URL "rtsp://cisco.com/content/CiscoCDS.mov," for the client IP address of 171.71.50.123, with an expiry delay of 120 seconds, a key ID owner of 1, a key ID number of 2, a key of kwnx90KGP, and the MD5 hash algorithm, enter the following:

python cds-ims-urlsign.py rtsp://cisco.com/content/CiscoCDS.mov 171.71.50.123 120 1 2 kwnx90KGP

The signed URL is the following:

```
rtsp://cisco.com/content/CiscoCDS.mov?IS=0&ET=1209422976&CIP=171.71.50.123&KO=1&KN=2&US=f0
8b56f46075813e44b2d4888628a471
```

Note The above signed URL is only an example. The hash algorithm generates a different message digest each time. For more information on the MD5 algorithm, see the IETF RFC 1321. For more information on the SHA-1 algorithm, see the IETF RFC 3174.

Importance of Device Synchronization

URL expiry time validation relies on the assumption that the clocks are synchronized on the server running the signing application and the Service Engines validating the URL. Use of Network Time Protocol (NTP) on all devices, including the device running the signing application or script, is highly recommended.

It is not sufficient to merely have the same local times on two devices while their time zones differ.

For example, the following two devices are not synchronized:

- Device 1:
  - Local Time: 11:00:59 PM, October 12, 2008
  - Time Zone: PST
- Device 2:
  - Local Time: 11:00:59 PM, October 12, 2008
  - Time Zone: EST

The following two devices are synchronized:

- Device 1:
  - Local Time: 11:00:59 AM, October 12, 2008
  - Time Zone: PST
- Device 2:
  - Local Time: 2:00:59 PM, October 12, 2008
  - Time Zone: EST

Understanding the Signing Procedure

To customize the URL signing script for your portal, or to write your own signing application in the platform and language of your choice, and still be able to validate URLs within the CDS, follow the steps explained in this section.

The URL signing script performs these steps when processing an unsigned URL:

1. Reads the version information from the script argument and removes the protocol from the URL if the version equals 2.

2. Checks if the URL already contains a query string.

   If the URL does not contain a query string, appends a question mark (?).

   If the URL does contain a query string, appends an ampersand (&).
3. Removes the port number in the URL, if one exists.
4. Appends the string **IS=0**. This string is for legacy support with some CDS components that use both internal (within CDS) and external (**portal**) signing mechanisms.
5. Appends the string **&ET=**.
6. Gets the current time in seconds since epoch (as an integer). Adds the expiry time in seconds as an integer and appends this integer.
7. Appends the string **&CIP=**.
8. Appends the requesting client IP address, using dotted decimal format.
9. Appends the string **&KO=**.
10. Appends the key ID owner corresponding to the key being used.
11. Appends the string **&KN=**.
12. Appends the key ID number corresponding to the key being used.
13. Appends the string **&US=**.
14. Stores this as url2; for example:

    "rtsp://cisco.com/content/CiscoCDS.mov?IS=0&ET=1209422976&CIP=171.71.50.123&KO=1&KN=2&US="
15. Generates an MD5 hash of the key being used.
16. Updates the generated hash with url2.
17. Converts the hash to its equivalent human readable hex digest; for example:

    f08b56f46075813e44b2d4888628a471
18. Appends the hex digest to url2. The URL signing is complete.

Public Key URL Signing

Release 2.4.1 and previous releases of the Internet Streamer CDS software support symmetric key URL signing. Symmetric key signing uses the same key to sign and validate the URL. Release 2.4.3 introduces asymmetric key signing, also known as public key URL signing. Asymmetric keys always have a key pair made up of a public key and private key. The private key is used for signing and the public key is used for validation.

Note The Public Key URL Signing feature is a Release 2.4.3 feature.

The public key URL signing supports Elliptic Curve (EC) keys and uses EC Digital Signature Algorithm (DSA), which is the EC equivalent of DSA for signature generation and signature validation.

Elliptic Curve Cryptography (ECC) has the following main advantages:

- Key size is small while still offering good security
- Key is easy to store
- Computation is faster than DSA or RSA

The signed URL of EC-DSA contains some clear text data (for example, client IP [CIP], expiry time [ET], and the US=DSA r and s values). For transport security, the Internet Streamer CDS software takes these tag values, converts them into hexadecimal values, then encrypts them using American Encryption Standard (AES) Counter (CTR) mode and 128-bits key size when the **url-signature** command **symmetric-key** option is configured by the user. The encrypted output is attached to the URL as base64 encoded data. Both hexadecimal and Base64 conversions produce URL-safe values, but Base64 produces smaller output, which is why AES encrypted output is converted to Base64.

How Public Key URL Signing Works with CDS

You can generate a pair of EC keys and write each key (public and private) into separate files (a public key file and a private key file) by using the Privacy Enhanced Mail (PEM) format. The **url-signature** command has new keyword options that provide a way to upload these files and associate them to a particular key owner and key number . If you want secure transmission of the CIP, ET, and US tag values, you can use the **symmetric-key** option of the **url-signature** command., which uses AES to encrypt the key.

The URL can be signed externally using the C binary. The signed URL is then sent to the CDS for signature validation. The CDS validates the signed URL by using the key owner (KO) and key number (KN) to look up the URL signature. You can also use the **url-signature** command to generate the URL signature, as well as validate it.

url-signature Command

The **url-signature** command creates a symmetric key or asymmetric key for the URL signature.

url-signature key-id-owner *key-id-owner* **key-id-number** *key-id-number* {**key** *key* | **public-key** *public-key* [**private-key** *private-key* [**symmetric-key**] | **symmetric-key**]}

Syntax Description

key-id-owner	Configure the owner for this key.
key-id-owner	Specify the ID for the owner of this key. Valid entries are from 1 to 32.
key-id-number	Configuring the ID number for this key.
key-id-number	Specify the ID for the number of this key. Valid entries are from 1 to 32.
key	Configure the symmetric encryption key for signing a URL.
key	Specify the encryption key. The maximum number of characters is 16. Spaces are not allowed.
public-key	Configure the public key for the specified key owner (KO) and key number (KN).
public-key	Specify the public key.
private-key	Optional. Configure the private key for the specified KO and KN.
private-key	Specify the private key.
symmetric-key	Optional. Use AES to encrypt the key.

Following is an example of generating and encrypting the public key and private key using the **url-signature** command:

```
(config)# url-signature key-id-owner 1 key-id-number 10 public-key http://1.1.1.1/ec_pub_key private-key
http://1.1.1.1/ec_pub_key symmetric-key
```

URL Signing C Program

To use the C binary to generate the signed URL, do the following:

Step 1 Get the following information from the client:

- URL
- Expiry time
- Client IP address
- Private key file
- Key owner
- Key number
- Symmetric key

Step 2 Construct the URL to be signed by removing the schema (the protocol of the URL, for example, HTTP) from the URL.

Step 3 Get the length of the constructed URL and add a tag “LENTOSIGN” before the US tag.

Step 4 Create a digest of the URL with the LENTOSIGN tag using SHA-1 without a key.

Step 5 Sign this digest with an EC private key.

Step 6 The signature contains two values, *r* and *s*. Convert them to Hexadecimal and add them to the signed URL.

Step 7 If an AES key is configured, convert the CIP, ET, and US tag values to hexadecimal values, and encrypt them using AES CTR 128 mode. The Initialization Vector (IV) used is 64 bits. The IV is encoded to base64 and added to the URL. After appending the IV to the URL , then append the encrypted data using AES CTR 128. This encrypted data should be encoded to base64 before adding it to URL.

Following is some sample output from C binary:

```
# ./public_key
 The usage is ./public_key <url> <client-ip> <expiry-delay-seconds> <key-id-owner>
<key-id-number> <private_key file> <Symmetric-Key>
The number of arguments is less than 8

# ./public_key rtsp://www.cisco.com/my.wmv 1.1.1.1 20000 1 2 test_priv ciscociscociscoc
Url : rtsp://www.cisco.com/my.wmv , Ko : 1 , KN : 2 Expiry_time : 20000
The Private Key read from file is : -----BEGIN EC PRIVATE KEY-----
MIIBUQIBAQQgNu8C5npnuJPzS+vUDLzbtVYHebXyd2fqI71cFIPky+uggeMwgeAC
AQEwLAYHKoZIzj0BAQIhAP////8AAAABAAAAAAAAAAAAAAAA////////////////
MEQEIP////8AAAABAAAAAAAAAAAAAAAA///////////////8BCBaxjXYqjqT57Pr
vVV2mIa8ZR0GsMxTsPY7zjw+J9JgSwRBBGsX0fLhLEJH+Lzm5WOkQPJ3A32BLesz
oPShOUXYmMKWT+NC4v4af5uO5+tKfA+eFivOM1drMV7Oy7ZAaDe/UfUCIQD/////
AAAAAP//////////vOb6racXnoTzucrC/GMlUQIBAaFEA0IABH7vJFy6si5SOY1E
40aByIjsFYuZ9eVuLyo1pyhnX0GINMfkLoJBT0KhJfah5zNuKRSi6V8NtUpaUc28
BYKqx6A=
-----END EC PRIVATE KEY-----

The ET time value is : 1248690812
The schema removed URL is : ://www.cisco.com/my.wmv
The URL to calculate LENTOSIGN is :
rtsp://www.cisco.com/my.wmv?SIGV=3&IS=0&CIP=1.1.1.1&ET=1248690812&KO=1&KN=2&US=
The URL LENTOSIGN value is : 79
The URL ready for sha1 sign without Key :
://www.cisco.com/my.wmv?SIGV=3&IS=0&CIP=1.1.1.1&ET=1248690812&KO=1&KN=2&LENTOSIGN=79&US=
The ECDSA Signed URL is :
rtsp://www.cisco.com/my.wmv?SIGV=3&IS=0&CIP=1.1.1.1&ET=1248690812&KO=1&KN=2&US=DSA=r:CFB03
EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E005668D:s:57ED0E8DF7E786C87E39177DD339
8A7FB010E6A4C0DC8AA71331A929A29EA24E
The base64 encoded IV is : cXgS7eo+sHc=
The ET value to be converted to Hex is : 1248690812The Hex converted ET string is :
0c304500080c
The IP to be converted to Hex is : 1.1.1.1Hex representation of IP Value is : 01010101
The length of SIG->r is : 64
The length of SIG->r is : 32
The constructed Hex string after adding SIG->r Tag representation :
01060c304500080c0204010101010332CFB03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E0
05668D
The length of SIG->s is : 64
The length of SIG->s is : 32
The data that will be encrypted is :
01060c304500080c0204010101010332CFB03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E0
05668D043257ED0E8DF7E786C87E39177DD3398A7FB010E6A4C0DC8AA71331A929A29EA24E
The AES Encrypted URL is :
rtsp://www.cisco.com/my.wmv?SIGV=3&IS=0&KO=1&KN=2&cXgS7eo+sHc=X+HOeJ6yKmUmmzLObphXZ98ttyj7
BaAeQF1hCaYBxwHgswiNAW+Uj+IHBLojKgxiCXULiPBmawF1czVKrvVmpvA8OoQ5ujJzpjYXeLLBGGSs3g==
```

C Program for URL Signing

The pseudo code for signature generation using the Public Key URL Signing feature has the following tasks:

Step 1 Make sure the C program accepts the following values as command line arguments to generate the signed URL:

- URL
- Client_IP
- Expiry_time
- KO
- KN
- Private Key file in PEM format
- Symmetric Key (for AES Encryption)

Step 2 After reading the arguments, check if the Private Key file can be read. If yes , run **fread** to read all the data .

Step 3 The URL given as the input argument to the program should then be passed to the function that can remove the schema (the protocol, for example, HTTP) from the URL. For example, http://www.cisco.com/index.html should be changed to ://www.cisco.com/index.html.

Step 4 Construct the URL with all the sign tags as follows:

```
snprintf(url_lentoadd , URL_MAX,
        "%s%c" "%s&" "%s" "&%s=%s" "&%s=%s" "&%s=%s" "&%s=%s" "&%s=",
        url,'?',
        sigv,"IS=0",
        "CIP",ip,
        "ET",cur_time,
        "KO",ko,
        "KN",kn,
        "US");
```

An example of the output follows:

```
http://www.cisco.com/index.html?SIGV=3&IS=0&CIP=1.1.1.1&ET=123456789&KO=1&KN=3&US=
```

```
In the above call snprintf , we construct the URL as above. The cur_time is calculated as
Epoch_time + expiry time .
```

Step 5 Calculate the length of the URL formed in Step 4.

For example, the length of URL http://www.cisco.com/index.html?SIGV=3&IS=0&CIP=1.1.1.1&ET=123456789&KO=1&KN=3&US is 82.

Step 6 Reconstruct the URL again with the new Tag LENTOSIGN added before the US tag as follows:

```
snprintf(url_to_sign, URL_MAX,
        "%s%c" "%s&" "%s" "&%s=%s" "&%s=%s" "&%s=%s" "&%s=%s" "&%s=%d" "&%s=",
        url_without_schema,'?',
        sigv,"IS=0",
        "CIP",ip,
        "ET",cur_time,
        "KO",ko,
        "KN",kn,
        "LENTOSIGN",len_to_add,
        "US");
```

An example of the output follows:

```
://www.cisco.com/index.html?SIGV=3&IS=0&CIP=1.1.1.1&ET=123456789&KO=1&KN=3&LENTOSIGN=82&US=
```

Note The value of LENTOSIGN is taken from Step 5. Also the URL does not have the schema (protocol).

Step 7 The URL is ready to be signed using SHA-1 without a key. An example of a URL ready for SHA-1 without a key follows: ://www.cisco.com/index.html?SIGV=3&IS=0&CIP=1.1.1.1&ET=123456789&KO=1&KN=3&LENTOSIGN=82&US=

The following APIs are used to sign the URL:

```
create_digest(sign,url_to_sign);
void create_digest(char *digest,char *data)
{
    memset(digest,0,sizeof(digest));
    int md_len;
    unsigned char digest1[20];
    EVP_MD_CTX md_ctx;
    EVP_MD_CTX_init(&md_ctx);
    EVP_DigestInit_ex(&md_ctx, EVP_ecdsa(),NULL);
    EVP_DigestUpdate(&md_ctx,data,strlen(data));
    EVP_DigestFinal_ex(&md_ctx,digest,&md_len);
}
```

Step 8 The private key is read from the file and EC_KEY is created using the following APIs:

```
EC_KEY *eckey=NULL;
    ECDSA_SIG *sig=NULL;
    BIO *out_priv=NULL;

out_priv=BIO_new_mem_buf(private_key,strlen(private_key));

eckey=(EC_KEY *)PEM_read_bio_ECPrivateKey(out_priv,NULL,NULL,NULL);
```

Step 9 The EC-DSA signature is created using the EC_KEY on the digest that was created in Step 7.

```
sig=ECDSA_do_sign(sign,20,eckey);
```

If the signature is successful, the *sig* parameter will contain r and s values. Extract the r and s values using the following commands:

```
BN_bn2hex(sig->r)
BN_bn2hex(sig->s)
```

Step 10 Add these signature values to the URL generated in Step 4.

```
strcat(url_lentoadd,"DSA=r:");
    strcat(url_lentoadd,BN_bn2hex(sig->r));
    strcat(url_lentoadd,":s:");
    strcat(url_lentoadd,BN_bn2hex(sig->s));
```

An example of the final EC-DSA signed URL follows:

```
http://www.cisco.com/index.html?SIGV=3&IS=0&CIP=1.1.1.1&ET=123456789&KO=1&KN=3&US=DSA=r:CF
B03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E005668D:s:57ED0E8DF7E786C87E39177DD
3398A7FB010E6A4C0DC8AA71331A929A29EA24E
```

Step 11 If transport security needs to be applied to the Tag values (CIP, ET, and US), use AES CTR 128 encryption. Convert them to hexadecimal format. Following is an example:

ET : 01 length hexadecimal_value of ET

ET will be 01 length hexadecimal (1248690812). The resulting value will be : 01 06 0c304500080c, which equals 01060c304500080c. The ET value is converted to hexadecimal by taking two digits at a time from the original ET, which in the example is 1248690812. Following are each two-digit conversion:

- 12 = 0c
- 48 = 30
- 69 = 45
- 08 = 00 08
- 12 = 0c

Each hexadecimal is four bits. The length is calculated based on the resulting values from the hexadecimal conversion. In the example, there are six 8-bit lengths: 0c 30 45 00 08 0c.

CIP : 02 length hexadecimal (IP)

CIP will be 02 length hexadecimal (1.1.1.1). The resulting value will be : 02 04 01010101, which equals 020401010101.

US : 03 length sig->r

US will be : 03 length (CFB03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E005668D). The resulting value will be : 03 32 CFB03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E005668D, which equals 0332CFB03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E005668D.

US : 04 length sig->s

US will be : 04 length (sig->s). The resulting value will be : 04 3257ED0E8DF7E786C87E39177DD3398A7FB010E6A4C0DC8AA71331A929A29EA24E, which is equal to 043257ED0E8DF7E786C87E39177DD3398A7FB010E6A4C0DC8AA71331A929A29EA24E.

The AES Key should be exactly16 bytes in length. It cannot be greater than 16 bytes or less than 16 bytes .

Step 12 Once the ET , CIP and US Tag values have been constructed in hexadecimal, they need to put together to send them to AES CTR encrypt API .

The values from Step 11 are the following:

```
01060c304500080c + 020401010101 +
0332CFB03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E005668D +
043257ED0E8DF7E786C87E39177DD3398A7FB010E6A4C0DC8AA71331A929A29EA24E
```

The data that will be encrypted using AES CTR 128 is the following:

```
01060c304500080c0204010101010332CFB03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E0
05668D043257ED0E8DF7E786C87E39177DD3398A7FB010E6A4C0DC8AA71331A929A29EA24E
```

The data that will be encrypted using AES CTR is converted to char * , so the length is 82 bytes.

Step 13 Before calling the APIs to perform the AES CTR encryption, the IV needs to be generated. The IV generated is 64 bits in length.

```
RAND_bytes(iv,8);
```

With the IV that is generated and the resultant data from Step 12, use the following APIs to encrypt the data using AES CTR mode.

```
AES_set_encrypt_key(sym_key,AES_BLOCK_SIZE*8,&keys);
AES_ctr128_encrypt();//Need to pass all the arguments
```

Step 14 Construct the AES Encrypted URL over EC-DSA signature as follows :

```
snprintf(final_aes,URL_MAX,
        "%s%c" "%s" "&%s=%s" "&%s=%s" "&%s=%s" "&%s=%s" "%s",
        url,'?',
        sigv,
        "IS","0",
        "KO",ko,
        "KN",kn,
        "US",
        base64_iv_p,
        base64_encode);
```

Note The IV generated is converted to Base64 format. It is attached to URL after the “US=” tag. The AES CTR encryption output is converted to Base64 format and appended to the URL after adding IV.

For example, if the Base64 encoded IV is : cXgS7eo+sHc=, and the AES CTR encryption output in Base64 format is : X+HOeJ6yKmUmmzLObphXZ98ttyj7BaAeQF1hCaYBxwHgswiNAW+Uj+IHBLojKgxiCXULiPBmawF1czVKrvVmpvA8OoQ5ujJzpjYXeLLBGGSs3g==

The final AES encrypted URL will be as follows:

```
http://www.cisco.com/index.html?SIGV=3&IS=0&KO=1&KN=3&US=
```

Step 15 Add the IV in Base64 format to the URL.

```
http://www.cisco.com/index.html?SIGV=3&IS=0&KO=1&KN=3&US=cXgS7eo+sHc=
```

Step 16 Finally, append the AES encryption output in Base64 format.

```
http://www.cisco.com/index.html?SIGV=3&IS=0&KO=1&KN=3&US=cXgS7eo+sHc=
X+HOeJ6yKmUmmzLObphXZ98ttyj7BaAeQF1hCaYBxwHgswiNAW+Uj+IHBLojKgxiCXULiPBmawF1czVKrvVmpvA8Oo
Q5ujJzpjYXeLLBGGSs3g==
```

These are the detailed steps that are used in generating a signed URL using Public Key URL Signing feature. This can be implemented completely using a C program.