

Acxis Dome Camera - Video Authentication

Mohammed Rahman

November 15, 2024

Contents

1	Setup	1
2	Network Storage	2
3	Video Signing	2
4	Framework Dependencies	3
4.1	Meson and ninja	3
4.2	OpenSSL	3
4.3	libcheck	3
4.4	GStreamer	4
5	Framework	4
6	Signer	5
7	Validator	5
8	Findings	7
8.1	Validating Non-Tampered Camera Footage	7
8.2	Validating Tampered Camera Footage	8

1 Setup

The setup documented follows the steps listed on the [datasheet](#). A Windows-based system was used to locate the IP address of the Axis Dome camera and setup shared storage, while a Linux-based system was used for the rest of the project.

The camera is currently configured with factory settings, including the default IP address. To locate the IP address, use either AXIS Utility or the Axis Device Manager. In this demonstration, AXIS Utility was used. When AXIS IP Utility is opened and the camera is connected, it should automatically detect the camera. The camera's IP address will be displayed on the left-hand side. Copy this address into a browser to access the camera software.

The details for the account are:

User: root

Password: Electronic1

To turn on video signing from this window, navigate to **Video** → **Stream** → **General**.

2 Network Storage

There are two types of storage available: network storage and SD cards. The former was chosen.

First, a shared folder will need to be created on the Windows machine (this can be done on Linux too), which can be done by following this [guide](#). Then, obtain the IP address of the machine hosting the shared folder. This can be done with:

Windows: ipconfig

Linux: ip a

Navigate to **System** → **Storage** → **Add Network Storage**. Enter the IP address of the host machine, along with the host's username, password, and the path to the shared folder. Leave all other settings at their default values.

3 Video Signing

AXIS cameras can utilise [AXIS Edge Vault](#) to sign videos. Ensure that the cameras in use have up-to-date firmware to enable video signing, as cameras straight from the box may not support this functionality. The process of video signing is outlined on pages 13 to 16 of the [datasheet](#).

4 Framework Dependencies

The signed-video-framework requires several dependencies, which are listed below and can be found on their [GitHub](#):

- meson
- ninja
- OpenSSL
- libcheck

To run the two applications, `signer` and `validator`, `GStreamer` will also be needed.

4.1 Meson and ninja

To install both `meson` and `ninja`, use the following commands:

```
$ pip install meson
$ pip install ninja
```

4.2 OpenSSL

To install `OpenSSL`, use the following commands:

```
$ git clone https://github.com/openssl/openssl
$ cd path/to/cloned/repo
$ ./Configure
$ make
$ make test
```

If the tests do not pass, you may need to run the commands by prepending `sudo`.

4.3 libcheck

To install libcheck, run the following command:

```
$ sudo apt-get install check
```

4.4 GStreamer

To install GStreamer, run the following command:

```
$ apt-get install libgstreamer1.0-dev  
libgstreamer-plugins-base1.0-dev  
libgstreamer-plugins-bad1.0-dev  
gstreamer1.0-plugins-base  
gstreamer1.0-plugins-good  
gstreamer1.0-plugins-bad  
gstreamer1.0-plugins-ugly  
gstreamer1.0-libav  
gstreamer1.0-tools  
gstreamer1.0-x  
gstreamer1.0-alsa  
gstreamer1.0-gl  
gstreamer1.0-gtk3  
gstreamer1.0-qt5  
gstreamer1.0-pulseaudio
```

5 Framework

Both the [signed-video-framework](#) and [signed-video-framework-examples](#) repositories were cloned under the same parent directory. All subsequent commands in the framework, as well as those for the `validator` and `signer` sections, were run in the parent directory.

The framework supports both H.264 and H.265 encoding. Depending on the encoding type, the appropriate format should be used in the signer and validator commands.

To compile the framework to the run the two apps, use the following commands:

```
$ meson --prefix $PWD/local_install signed-video-framework
```

```
build
$ ninja -C build
$ meson install -C build
$ ninja -C build test
```

The unit tests should all pass if done correctly.

6 Signer

The commands listed below used to compile the validator can also be found on the [AXIS Signer GitHub](#).

If the camera does not automatically perform the signing, the signer step can be done manually. To do this, you must first compile the signer by following the commands provided below:

```
$ export GST_PLUGIN_PATH=$PWD/my_installs
$ meson --prefix $PWD/my_installs -Dsigner=true signed-
  video-framework-examples build_signer
$ meson install -C build_signer
```

The `GST_PLUGIN_PATH` environment variable must be set to the local installation directory, as the signer application is implemented as a GStreamer element.

If the signer application is compiled successfully, it will be located in `local_install/bin/signer`.

To run the signer on a test file, first copy the test file into the same directory where the command is executed. In this case, the file should be placed in the parent directory:

```
$ ./local_install/bin/signer -c h264 test_h264.mp4
```

If the signer is successful, a new file with the prefix `signed_` should be generated.

7 Validator

The commands listed below used to compile the validator can also be found on the [AXIS Validator GitHub](#).

```
$ meson --prefix $PWD/local_install -Dvalidator=true signed-  
video-framework-examples build_validator  
$ meson install -C build_validator
```

If the validator application is compiled successfully, it will be located in `local_install/bin/validator`.

To run the validator on a test file, first copy the test file into the same directory where the command is executed. In this case, the file should be placed in the parent directory:

```
$ ./local_install/bin/validator -c h264 test_h264.mp4
```

The results can be viewed in the `validation_results` text file produced.

8.1 Validating Non-Tampered Camera Footage

[illegible]

As shown in Figure 1, the video is validated, indicating that both the public key is valid and the video has not been tampered with. This aligns with our expectations. However, it also shown that the public key cannot be validated. This means the validator is unable to confirm the authenticity of the public key itself. As stated by the developers on GitHub, this is outside the scope of the framework’s purpose:

Possible solutions to address this issue include validating the public key through a Certificate Authority or implementing a lookup system to verify

whether the public key in question is included in a trusted collection of public keys.

8.2 Validating Tampered Camera Footage

A PNG image was inserted into the signed video from the above section. Figure 2 below shows the result of running the validator on this altered signed video captured by the AXIS Dome camera.

[illegible]

Figure 2: Running the validator on a tampered signed video from the AXIS Dome camera.

Once again, the public key could not be validated. The validator correctly rejected the video, but it was not marked as invalid. Instead, the video was labeled as "not signed" rather than "invalid".

I believe this issue arises from the video being altered. In the original signed video, there were 8 signed GOPs (Group of Pictures), whereas the altered video shows only 3 unsigned GOPs. However, I don't see this as a problem because all the videos produced by the AXIS Dome camera are

expected to be signed. If the validator returns "unsigned," it clearly indicates that the video has been tampered with.