

Axis Dome Camera - Video Authentication

Mohammed Rahman

December 10, 2024

Contents

1	Setup	2
2	Network Storage	2
3	Video Signing	3
4	Framework Dependencies	3
4.1	Meson and ninja	3
4.2	OpenSSL	3
4.3	libcheck	4
4.4	GStreamer	4
5	Framework	4
6	Signer	5
7	Validator	5
8	Product Information	6
9	Public Key Validation	8
9.1	Validation and Verification Process	8
9.2	Signing Manually with the Signer Software	15
9.3	Ensuring Secure Public Key Validation	15
10	Findings	16
10.1	Validating Non-Tampered Camera Footage	16
10.2	Validating Tampered Camera Footage	17

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 Product Information

Figure 1 below illustrates the product information as displayed by the validator.

```
Product Info  
-----  
Hardware ID:      931.13  
Serial Number:    B8A44F46F6B2  
Firmware version: 11.10.61  
Manufacturer:     Axis Communications AB  
Address:          www.axis.com  
-----
```

Figure 1: Product information displayed by the validator.

The validator retrieves this information from the metadata, which is stored within the MKV file, as demonstrated in Figure 2.


```

H0F ! .J..m.k. ) . h ...I .&.4.... .. ! ...D..`!...$.Eb{.....F.~..... ---
--BEGIN CERTIFICATE----- MIIcMTCcAh6gAwIBAgIUCzUVEUJzyvyoH4BKgQkryGga6tswCgY
IKoZiZj0EAwIw ajEfMB0GA1UEChMWQXhpcyBDb21tdW5pY2F0aW9ucyBBQjEYMBYGA1UECjMPQX
hp cyBFZGdlIFZhdWx0MS0wKwYDVQDEyRBeGlzIEVkb2UgVmF1bHQgQXR0ZXN0YXRp b24gQ0Eg
RUNDIDewHhcNMjEwOTUwMTM1NjAwWhcNMzEwOTUwMTM1NjAwWjCBuTEL MAKGA1UEBhMCU0UxDTA
LBgNVBACTBEx1bmQxH2AdBgNVBAoTFkF4aXMgQ29tbXVv aWNhdGlvbnMgQUIxGDAWBgNVBAsTD0
F4aXMgRWRnZSBWYXVsdDFJMEcGA1UEAxNA QXhpcyBFZGdlIFZhdWx0IEF0dGVzdGF0aW9uIDA0M
DA1MDAxNTMzRkZERTJGQjQz Q0UwNDNEN0E2QTZGNzQ4MDEVMBMGA1UEBRMMQjhhBNDRGNDZGNkIy
MFkwEwYHKoZI zj0CAQYIKoZIzj0DAQcDQgAEVMT3a3l1hNgE1iJkWRzzfndoNEhGfRkQkdVcCA5
z mr3u+kR7UcSR9sU1bKGDnqyKG0/WLW5tFlKjnA1+P4ICaNSMFAwHQYDVRO0BBYE FNOLmG5Kf
D1D2NroI/dLmSLtQ+Q5MA4GA1UdDwEB/wQEAWIGwDAFBgNVHSMEGDAW gBREzSmfIeeg22w1ODia
5Pg33kUZKTAKBggqhkJOPQDAGNpADBMAjEAsx4idSOD rAy8x9Y6sCGouopiBr0iWX38+499Qqa
rdie18v2NrIH19TilozBqz/vWAjEAhlcq N8ss2YhBeARr42/mRnHW11ZHy0yl76K0oYmna65Dyy
LT5y0KyBljYShwJ10v -----END CERTIFICATE----- -----BEGIN CERTIFICATE----- MII
CjjCCAE+gAwIBAgIQMALJ2gXvfCgj1rpCsrMJDABgqhkJOPQDDBDBcMR8w HQYDVQKQEXZBeG
lzIENvW11bmljYXRpb25zIEFCMRGwFgYDVQLEw9BGeGlzIEVkb2UgVmF1bHQxH2AdBgNVBAMTF
kF4aXMgRWRnZSBWYXVsdCBDBQSBFQ0MwIBcNMjAx MDI2MDg1ODM0WhgPMjA1NTA2MDExMjAwMDBa
MGoxH2AdBgNVBAoTFkF4aXMgQ29t bXVuaWNhdGlvbnMgQUIxGDAWBgNVBAsTD0F4aXMgRWRnZSB
WYXVsdDEtMCsGA1UE AxMkQXhpcyBFZGdlIFZhdWx0IEF0dGVzdGF0aW9uIENBIEVDQyAxMHYwEA
YHKoZI zj0CAQYFK4EEACIDYgAEligD2YyLfvMXgra74Q1K7/vam+yweU1LF5uY+qi9AthJ y7jP
V7GgIk+b/CACIinriKhOr/vUrgJuy8xN50xnC61hZvC0TkGlx1kt4m92cmFE 4Y1MgwMDbHPobyw
vxvEko2YwZDASBgNVHRMBAf8ECDAGAQH/AgEAMA4GA1UdDwEB /wQEAWIBBjAdBgNVHQ4EFgQURM
0pnyHnoNtsJtG4muT4N95FGSkwHwYDVROjBBgw FoAUkFoBqsPSSKuY+ZG8J2szqS3RcKewCgYIK
oZiZj0EAwQDgYwAMIGIAkIA9Pwn Yo+6a7KYC7tBsmPQtbcLtmRmKGR+XjHWGVkNH36JNuyjmNvU
DjVxNNbAsVy6y1g3 MLy0IC19hr2RRy1FP9QCQgCf0753SWnxgq78EPDiUg0Tn53/sintVZkUtlLU
U95y9 GZY5sFpisz7gdA17PD5tIje6eSM30Rxy2X+ufbZQm7D5w== -----END CERTIFICATE-
----- M G0E 8. P.O.. .. ;;. ..W. . ...s!. ! ...}z.= .... -...E.. .V

```

Figure 4: The CA Certificates found in Hex Fiend.

9 Public Key Validation

9.1 Validation and Verification Process

The Signed Video Framework ensures the public key is validated each time before use, as specified in lines 46–48 of the [source code](#).

The framework validates the public key by verifying it against a trusted certificate chain, ensuring its authenticity through a series of signed certificates from a trusted certificate authority (CA). Specifically, the framework uses a static trusted Axis root CA certificate, which is defined on line 51 of the [source code](#).

As demonstrated in the previous section, the signed video metadata includes two certificates: the intermediate certificate, which is signed by the Axis root CA certificate. And the attestation certificate, which contains the public key used to verify the signed video, this is signed by the intermediate certificate.

The certificates were successfully decoded using `OpenSSL`. To achieve this,

I stored each certificate in a `.pem` file and used the following command to decode the certificate:

```
$ openssl x509 -in file.pem -text -noout
```

The `-in` option specifies the input file, `-text` displays the decoded certificate in a human-readable format on the command line, and `-noout` suppresses the output of the certificate's binary content. Shown below is the decoded attestation certificate.

```
Version: 3 (0x2)
Serial Number:
    0b:35:15:11:42:73:ca:fc:a8:1f:80:4a:81:09:2b:c8:68:1a:ea
    :db
Signature Algorithm: ecdsa-with-SHA256
Issuer: O=Axis Communications AB, OU=Axis Edge Vault,
        CN=Axis Edge Vault Attestation CA ECC 1
Validity
    Not Before: Sep 10 13:56:00 2021 GMT
    Not After : Sep 10 13:56:00 2031 GMT
Subject: C=SE, L=Lund, O=Axis Communications AB, OU=Axis
        Edge Vault,
        CN=Axis Edge Vault Attestation 04005001533FFDE2F
        B43CE043D7A6A6
        F7480, serialNumber=B8A44F46F6B2
Subject Public Key Info:
    Public Key Algorithm: id-ecPublicKey
    Public-Key: (256 bit)
    pub:
        04:54:c4:f7:6b:79:75:84:d8:04:d6:22:64:c1:1c:
        f3:7e:77:68:34:48:46:7d:18:50:91:d5:5c:08:0e:
        73:9a:bd:ee:fa:44:7b:51:c4:91:f6:c5:35:6c:a1:
        83:9e:ac:a4:18:ef:d6:2e:55:b9:b4:59:4a:8e:70:
        35:f8:fe:08:09
    ASN1 OID: prime256v1
    NIST CURVE: P-256
X509v3 extensions:
    X509v3 Subject Key Identifier:
        D3:8B:98:6E:4A:7C:3D:43:D8:DA:E8:23:F7:4B:99:29:53:
        43:E4:39
```

```
X509v3 Key Usage: critical
    Digital Signature, Non Repudiation
X509v3 Authority Key Identifier:
    44:CD:29:9F:21:E7:A0:DB:6C:25:38:38:9A:E4:F8:37:DE:
    45:19:29
Signature Algorithm: ecdsa-with-SHA256
Signature Value:
    30:66:02:31:00:b3:1e:22:75:23:83:ac:0c:bc:c7:d6:3a:b0:
    21:a8:ba:8a:62:06:b3:a2:59:7d:fc:fb:8f:7d:41:06:ab:76:
    27:b5:f2:fd:8d:ac:81:f5:f5:38:a5:a3:30:6a:cf:fb:d6:02:
    31:00:86:57:2a:37:cb:2c:d9:88:41:78:04:6b:e3:6f:e6:46:
    71:d6:d7:56:47:cb:4c:a5:ef:a2:8e:a1:83:27:6b:ae:43:cb:
    22:d3:e7:23:8a:c8:19:63:61:28:70:27:53:af
```

Version: The version of the X.509 certificate. X.509 is a widely adopted standard for digital certificates used to verify public keys and establish trust in secure communications.

Serial Number: A unique identifier assigned to the certificate by the Certificate Authority (CA). This distinguishes this certificate from others issued by the same CA.

Algorithm: The algorithm used to sign the certificate, combining the Elliptic Curve Digital Signature Algorithm (ECDSA) with SHA-256 hashing for security.

Issuer Information: Identifies the Certificate Authority (CA) that issued the certificate:

- **O:** Organisation (Axis Communications AB)
- **OU:** Organisational Unit (Axis Edge Vault)
- **CN:** Common Name (Axis Edge Vault Attestation CA ECC 1)

As observed below in the intermediate certificate, the issuer information presented here corresponds directly to the subject information of the intermediate certificate.

Not Before/After: The time period during which the certificate is valid. After the "Not After" date, the certificate is considered expired.

Subject Information: The subject information outlines details about

the certificate holder. Key fields such as the Organisation, Organisational Unit, and Common Name are propagated to certificates issued by this certificate, where they appear as issuer information

- **C:** Country (SE = Sweden)
- **L:** Locality
- **O:** Organisation
- **OU:** Organisational Unit
- **CN:** Common Name, including an identifier for the specific device or application. The long number present in the CN is the chip ID of the axis camera.
- **serialNumber:** Serial number of the Axis Camera used for signing.

Public Key Algorithm: The type of public key algorithm used (Elliptic Curve Public Key).

Public-Key Size: The size of the public key, 256 bits, which is suitable for strong encryption.

Public Key: This is the public key in hexadecimal format, which is stored within the certificate. It can be used by certificates issued by this certificate to verify the validity of their signatures.

ASN1 OID: prime256v1 Specifies the elliptic curve used in the public key.

Subject Key Identifier: This is typically a hash of the public key. This identifier makes it easier to find and manage certificates, especially in a large Public Key Infrastructure (PKI). This is inherited by certificates issued by this certificate under the **Authority Key Identifier** field.

Key Usage: Specifies allowed uses of the certificate, such as digital signatures and non-repudiation.

Authority Key Identifier: Identifies the public key used by the issuing CA.

Algorithm: Specifies the algorithm used to sign the certificate's contents.

Signature Value: The actual digital signature generated by the CA to

verify the authenticity of the certificate.

The decoded intermediate and root certificates are also shown below.

```
Version: 3 (0x2)
Serial Number:
    30:02:c9:da:05:ef:7c:28:23:27:5a:e9:71:24:66:24
Signature Algorithm: ecdsa-with-SHA512
Issuer: O=Axis Communications AB, OU=Axis Edge Vault,
        CN=Axis Edge Vault CA ECC
Validity
    Not Before: Oct 26 08:58:34 2020 GMT
    Not After : Jun  1 12:00:00 2055 GMT
Subject: O=Axis Communications AB, OU=Axis Edge Vault,
        CN=Axis Edge Vault Attestation CA ECC 1
Subject Public Key Info:
    Public Key Algorithm: id-ecPublicKey
    Public-Key: (384 bit)
    pub:
        04:96:28:03:d9:8c:8b:7e:f3:17:82:b6:bb:e1:0d:
        4a:ef:fb:da:9b:ec:b0:79:42:0b:17:9b:98:fa:a8:
        bd:02:d8:49:cb:b8:cf:57:b1:a0:22:4f:9b:fc:20:
        02:22:29:eb:88:a8:4e:af:fb:d4:ae:02:6e:cb:cc:
        4d:e7:4c:67:0b:ad:61:66:f0:b4:4e:41:a5:c6:59:
        2d:e2:6f:76:72:61:44:e1:8d:4c:83:03:03:6c:73:
        e8:6f:2c:2f:c6:f1:0a
    ASN1 OID: secp384r1
    NIST CURVE: P-384
X509v3 extensions:
    X509v3 Basic Constraints: critical
        CA:TRUE, pathlen:0
    X509v3 Key Usage: critical
        Certificate Sign, CRL Sign
    X509v3 Subject Key Identifier:
        44:CD:29:9F:21:E7:A0:DB:6C:25:38:38:9A:E4:F8:37:DE:
        45:19:29
    X509v3 Authority Key Identifier:
        90:5A:01:AA:C3:D2:48:AB:98:F9:91:BC:27:6B:33:A9:2D:
        D1:70:A1
Signature Algorithm: ecdsa-with-SHA512
```

Signature Value:

30:81:88:02:42:00:f4:f5:a7:62:8f:ba:6b:b2:98:0b:bb:41:
b2:6a:50:b5:b7:0b:b4:ca:cc:90:64:7e:5e:31:d6:19:59:0d:
1f:7e:89:36:ec:a3:98:db:d4:0e:35:71:34:d6:c0:b1:5c:ba:
cb:58:37:30:bc:8e:20:2d:7d:86:bd:91:47:2d:45:3f:d4:02:
42:00:9f:3b:be:77:49:69:f1:82:ae:fc:10:f0:e2:52:0d:13:
9f:9d:ff:b2:29:ed:55:99:14:b4:b5:14:f7:9c:bd:19:96:39:
b0:5a:62:b3:3e:e0:74:09:7b:3c:3e:6d:22:37:ba:79:23:37:
39:1c:72:a7:65:fe:b9:f6:d9:42:6e:c3:e7

Version: 3 (0x2)

Serial Number: 1 (0x1)

Signature Algorithm: ecdsa-with-SHA512

Issuer: O=Axis Communications AB, OU=Axis Edge Vault,
CN=Axis Edge Vault CA ECC

Validity

Not Before: Oct 26 08:43:13 2020 GMT

Not After : Oct 26 08:43:13 2035 GMT

Subject: O=Axis Communications AB, OU=Axis Edge Vault,
CN=Axis Edge Vault CA ECC

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (521 bit)

pub:

04:01:26:7e:3c:51:89:3a:ef:8c:b6:68:97:d8:06:
dd:80:94:c5:ca:16:8a:16:f3:d8:bd:fe:ea:ca:75:
db:42:3e:28:0f:ed:2e:16:08:0c:c6:a2:b3:7d:8c:
d2:d0:2a:95:de:55:0e:8d:45:5a:f8:ba:90:45:6d:
a9:09:f8:df:ea:ce:ce:01:fc:5e:45:e8:94:ff:90:
14:d4:9d:29:43:f2:25:ac:27:33:80:7d:fe:33:d9:
e8:e0:39:6c:6e:d1:d4:23:4a:98:fe:09:a6:46:16:
aa:b0:f6:da:18:dc:01:5f:47:7f:f8:73:e5:32:b6:
a5:ce:48:04:72:31:67:23:1c:9e:50:5c:db

ASN1 OID: secp521r1

NIST CURVE: P-521

X509v3 extensions:

X509v3 Subject Key Identifier:

90:5A:01:AA:C3:D2:48:AB:98:F9:91:BC:27:6B:33:A9:
2D:D1:70:A1

X509v3 Authority Key Identifier:

```
90:5A:01:AA:C3:D2:48:AB:98:F9:91:BC:27:6B:33:A9:
2D:D1:70:A1
X509v3 Basic Constraints: critical
CA:TRUE, pathlen:1
X509v3 Key Usage: critical
Certificate Sign, CRL Sign
Signature Algorithm: ecdsa-with-SHA512
Signature Value:
30:81:87:02:41:51:fc:22:04:ad:13:21:12:5e:6d:69:bd:fe:
7b:0d:00:49:23:6f:16:a8:e3:4a:1e:31:48:3e:23:79:83:36:
be:e0:b8:d2:54:a3:8b:a0:39:f4:e8:1a:99:e6:dd:02:ce:84:
42:73:c8:7a:8f:04:74:c2:87:11:1d:e5:b1:95:51:6f:02:42:
00:d1:fd:90:2b:e3:4e:4e:86:4e:64:a4:cf:e0:9b:31:0b:3f:
e7:d8:0d:97:ee:4c:5b:ad:c3:74:bc:1c:c6:12:bc:c5:9b:4c:
2c:cd:fa:a9:ea:f8:65:75:82:84:92:ea:f8:d5:af:de:44:a4:
3c:42:54:8c:64:28:eb:ee:c6:1c:95:0e
```

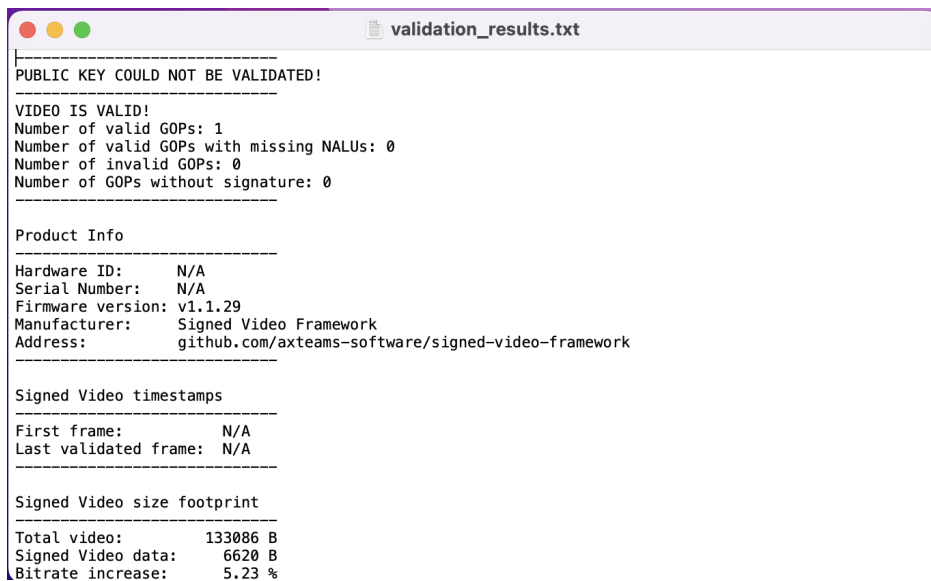
As observed in the certificates, the attestation certificate's issuer points to the subject information of the intermediate certificate, and the intermediate certificate's issuer points to the subject information of the root certificate. Additionally, the **Authority Key Identifier** for the root certificate matches the **Subject Key Identifier**, as the root certificate is inherently trusted. Therefore, the root certificate must be securely stored, as the entire process of validating the public key of the attestation certificate relies on the assumption that the root certificate is valid.

Furthermore, The signature within the attestation certificate, is verified using the public key of the intermediate certificate. If this verification is successful, the signature of the intermediate certificate is then verified against the Axis root CA certificate. If the intermediate certificate is valid, this implies the attestation certificate is also valid. This process is detailed in the [source code](#), specifically in the function `verify_certificate_chain` at line 145.

If the attestation certificate is successfully validated, it confirms that the public key contained within the certificate has been validated by the Axis root CA certificate. The transmitted public key in the metadata is then compared to the signature in the attestation certificate, as shown in the function `verify_axis_communications_public_key` on line 372 of the [source code](#). If they match, this indicates that the transmitted public key is the same as the validated public key within the attestation certificate.

9.2 Signing Manually with the Signer Software

An unsigned video produced by the Axis Camera was signed using the signer software provided on the Axis Github. Shown below is the metadata of the signed video found in the `validation_results.txt` file.



```
validation_results.txt
-----
PUBLIC KEY COULD NOT BE VALIDATED!
-----
VIDEO IS VALID!
Number of valid GOPs: 1
Number of valid GOPs with missing NALUs: 0
Number of invalid GOPs: 0
Number of GOPs without signature: 0
-----

Product Info
-----
Hardware ID:    N/A
Serial Number:  N/A
Firmware version: v1.1.29
Manufacturer:   Signed Video Framework
Address:        github.com/axteams-software/signed-video-framework
-----

Signed Video timestamps
-----
First frame:    N/A
Last validated frame: N/A
-----

Signed Video size footprint
-----
Total video:    133086 B
Signed Video data: 6620 B
Bitrate increase: 5.23 %
```

Figure 5: Metadata found in the `validation_results.txt` file

Important to note is that the serial number and hardware ID are absent, as this was not signed by a physical camera. Additionally, the address points to the GitHub repository.

Furthermore, when opening the signed MKV file in **Hex Fiend**, only the public key was found, with no traces of CA certificates. This is most likely due to the requirement for the intermediate certificate, which must be issued by the root certificate. The intermediate certificate contains a signature signed by the private key of the root certificate, and this private key cannot be stored in the GitHub repository.

9.3 Ensuring Secure Public Key Validation

This piece of code, which validates the public key using certificates, seems to contradict the developers' statement:

”The framework signals the video as authentic given that the Public key can be trusted. Validating the Public key is out of scope.”

Trusting the public key inherently implies that the corresponding private key must also be trusted. However, the private key is only used during the signing process, which occurs within the Axis camera itself. Therefore, the public key can be trusted if we can verify that it was signed by the trusted Axis camera. This verification can be accomplished by comparing the serial number and chip ID in the certificate metadata to those of the camera we trust.

I believe that each Axis camera has a unique intermediate certificate generated by the root certificate, which can be used to verify if a signed video matches the correct camera. To confirm this, I would need to test this theory with a different Axis camera.

Therefore, if a malicious user wanted to alter the video, they would need to remove all GOPs and metadata to revert the video to its unsigned state, then re-sign the video using the same chip ID, serial number, and public key of the original video. This can only be done if they have access to the private key used to sign the video, which would only be possible if the private key were somehow compromised.

10 Findings

10.1 Validating Non-Tampered Camera Footage

Figure 6 below demonstrates running the validator on a signed video captured by the AXIS Dome camera, the command used is also shown.

```
$ ./local_install/bin/validator -c h264 20241107_1223_86DE.mkv
```

The video validation confirms two key points: the public key corresponds accurately to the private key, and the video remains unaltered, ensuring its integrity. 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.

This was an unusual issue encountered while running the validator. On the Linux machine onsite, the public key was successfully validated. However,


```
validation_results.txt

PUBLIC KEY COULD NOT BE VALIDATED!

=====
VIDEO IS VALID!
Number of valid GOPs: 1
Number of valid GOPs with missing NALUs: 0
Number of invalid GOPs: 0
Number of GOPs without signature: 0
=====

Product Info
=====
Hardware ID:      931.13
Serial Number:    B8A44F46F6B2
Firmware version: 11.10.61
Manufacturer:     Axis Communications AB
Address:          www.axis.com
=====

Signed Video timestamps
=====
First frame:      Thu 2024-11-07 12:23:43 UTC
Last validated frame: Thu 2024-11-07 12:23:43 UTC
=====

Signed Video size footprint
=====
Total video:      48402 B
Signed Video data: 2440 B
Bitrate increase: 5.31 %
** Message: 10:07:11.275: GST pipeline: filesrc location="20241107_122343_86DE.mkv" ! matroskademux ! h264parse ! video/x-h264,
stream-format=byte-stream,alignment=(string)nal ! appsink name=validatorsink
** Message: 10:07:11.417: Latest authenticity result:
    nalus : _vvi_P_P_P_P_P_P_S
    valid : _P_P_P_P_P_P_P_
** Message: 10:07:11.418: Validation performed with Signed Video version v1.1.29
** Message: 10:07:11.418: Signing was performed with Signed Video version v1.1.29
** Message: 10:07:11.418: Validation complete. Results printed to 'validation_results.txt'.
```

Figure 6: Running the validator on a signed video from the AXIS Dome camera.

when running the validator on my personal Mac machine, the public key validation failed. I was unable to fix this issue. However, I believe it is not significant. If we can establish trust in the public key, the validator's determination that the video is authentic inherently confirms the validity of the public key.

10.2 Validating Tampered Camera Footage

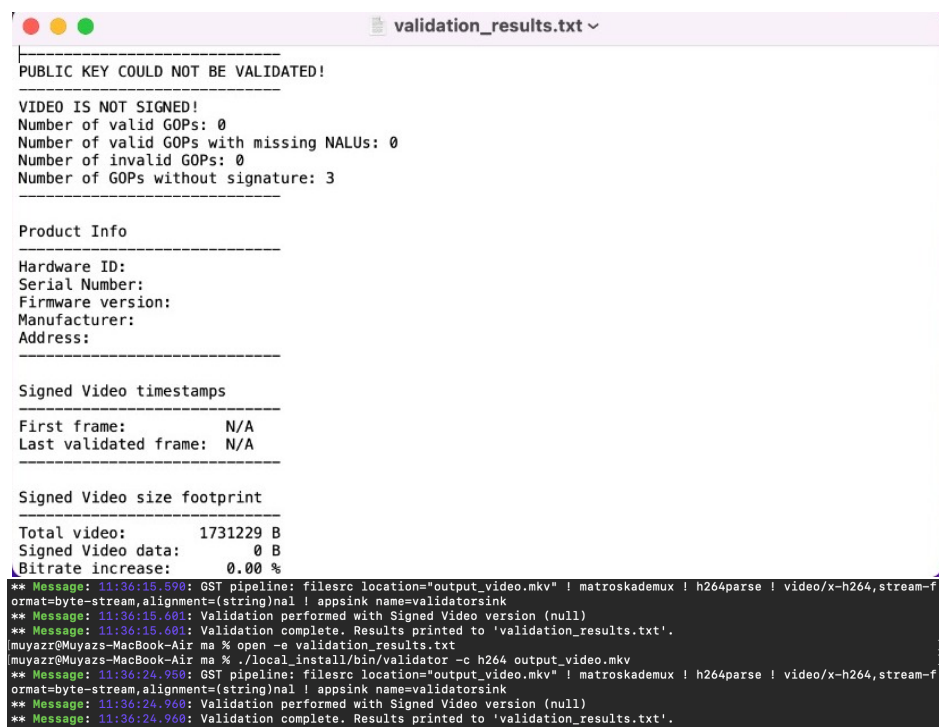
A PNG image was inserted into the signed video from the above section. This was accomplished using `ffmpeg` with the following command:

```
$ ffmpeg -i input_video.mkv -i image.png -filter_complex
"overlay=x=10:y=10" -c:a copy output_video.mkv
```

This `ffmpeg` command performs the following actions: `-i input_video.mkv` specifies the input video file, `-i image.png` specifies the input image to be

overlaid on the video, `-filter_complex "overlay=x=10:y=10"` applies the overlay filter to position the image 10 pixels from the top-left corner of the video, `-c:a copy` copies the audio stream without re-encoding to preserve its original quality, and `output_video.mkv` defines the output file where the video with the overlaid image and original audio will be saved.

Figure 7 below shows the result of running the validator on this altered signed video captured by the AXIS Dome camera.



```
validation_results.txt
-----
PUBLIC KEY COULD NOT BE VALIDATED!
-----
VIDEO IS NOT SIGNED!
Number of valid GOPs: 0
Number of valid GOPs with missing NALUs: 0
Number of invalid GOPs: 0
Number of GOPs without signature: 3
-----

Product Info
-----
Hardware ID:
Serial Number:
Firmware version:
Manufacturer:
Address:
-----

Signed Video timestamps
-----
First frame:      N/A
Last validated frame: N/A
-----

Signed Video size footprint
-----
Total video:      1731229 B
Signed Video data: 0 B
Bitrate increase: 0.00 %

** Message: 11:36:15.590: GST pipeline: filesrc location="output_video.mkv" ! matroskademux ! h264parse ! video/x-h264,stream-f
ormat=byte-stream,alignment=(string)nal ! appsink name=validatorsink
** Message: 11:36:15.601: Validation performed with Signed Video version (null)
** Message: 11:36:15.601: Validation complete. Results printed to 'validation_results.txt'.
muyazr@Muyazs-MacBook-Air ma % open -e validation_results.txt
muyazr@Muyazs-MacBook-Air ma % ./local_install/bin/validator -c h264 output_video.mkv
** Message: 11:36:24.950: GST pipeline: filesrc location="output_video.mkv" ! matroskademux ! h264parse ! video/x-h264,stream-f
ormat=byte-stream,alignment=(string)nal ! appsink name=validatorsink
** Message: 11:36:24.960: Validation performed with Signed Video version (null)
** Message: 11:36:24.960: Validation complete. Results printed to 'validation_results.txt'.
```

Figure 7: 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.

A snippet of signed video from the 10 second mark to the 20 second mark was taken out, this was achieved with the following commands below:

```
$ ffmpeg -i 20241112_103959_22E6.mkv -ss 00:00:00 -to  
00:00:10 -c copy part1.mkv  
$ ffmpeg -i 20241112_103959_22E6.mkv -ss 00:00:20 -to  
00:01:00 -c copy part2.mkv  
$ echo "file 'part1.mkv'" > filelist.txt  
$ echo "file 'part2.mkv'" >> filelist.txt  
$ ffmpeg -f concat -safe 0 -i filelist.txt -c copy  
output.mkv
```

The result of running the validator is shown below.

The validator correctly marks the video as invalid. Three valid GOPs are detected, which likely correspond to the portion of the video before it was trimmed. However, after the trim, the subsequent GOPs are invalid because they depend on the previous GOPs, as demonstrated by the four invalid GOPs following the trim.


```
** Message: 11:19:53.814: GST pipeline: filesrc location="part1.mkv" ! matroskademux ! h264parse ! video/x-h264,stream-format=b
yte-stream,alignment=(string)nal ! appsink name=validatorsink
** Message: 11:19:53.848: Latest authenticity result:
  nalus : _vvi_P_P_P_P_P_S
  valid  : __P_P_P_P_P_P_
** Message: 11:19:53.848: Validation performed with Signed Video version v1.1.29
** Message: 11:19:53.848: Signing was performed with Signed Video version v1.1.29
** Message: 11:19:53.848: Validation complete. Results printed to 'validation_results.txt'.
```

validation_results.txt

```
-----
PUBLIC KEY COULD NOT BE VALIDATED!
-----

VIDEO IS VALID!
Number of valid GOPs: 4
Number of valid GOPs with missing NALUs: 0
Number of invalid GOPs: 0
Number of GOPs without signature: 0
-----

Product Info
-----
Hardware ID:      931.13
Serial Number:    B8A44F46F6B2
Firmware version: 11.10.61
Manufacturer:     Axis Communications AB
Address:          www.axis.com
-----

Signed Video timestamps
-----
First frame:      Tue 2024-11-12 10:39:59 UTC
Last validated frame: Tue 2024-11-12 10:40:11 UTC
-----

Signed Video size footprint
-----
Total video:      2359590 B
Signed Video data: 15896 B
Bitrate increase: 0.68 %
```

Figure 10: Running the validator on a tampered signed video from the AXIS Dome camera.

only 4 valid GOPs remain. This suggests that while some GOPs were removed, the validator still passes the remaining GOPs as valid. I believe the previous test failed because it involved removing GOPs between other GOPs, whereas in this test, only the initial GOPs were trimmed.

The developers state:

”Full tampering coverage is not guaranteed. There are still various tampering scenarios that are not handled and also, authenticity is a matter of trust.”

This may suggest that this type of tampering is not covered by the framework.

22