D7.7 First Version of the IoT Key Distribution Prototype

Document Identification			
Status	Final	**Due Date**	31/10/2019
Version	1.0	**Submission Date**	31/10/2019

Related WP	WP7	**Document Reference**	D7.7
Related Deliverable(s)	D7.1, D7.2, D7.3, D7.5	**Dissemination Level(*)**	PU
Lead Participant	Kudelski (KUD)	**Lead Author**	Johan Cattin (Kudelski)
Contributors	Kudelski	**Reviewers**	Kimmo Järvinen (UH) Norman Scaife (WALLIX)

Keywords:
Preliminary Version Demonstrator Report, technical specification, use-case, camera, IoT, crypto API

This document is issued within the frame and for the purpose of the FENTEC project. This project has received funding from the European Union's Horizon2020 under Grant Agreement No. 780108. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

Document Information

List of Contributors	
Name	Partner
Johan Cattin	Kudelski
Aymeric Genet	Kudelski
Luca Gradassi	Kudelski
Daniel Nunes Silva	Kudelski
Yolan Romailler	Kudelski

Document History			
Version	Date	Change editors	Changes
0.1	30/07/2019	Kudelski	Creation
0.2	10/10/2019	Kudelski	First draft
0.3	22/10/2019	Kudelski	Published for internal review
0.4	25/10/2019	Kudelski	WALLIX comments addressed
0.5	29/10/2019	Kudelski	UH comments addressed
1.0	31/10/2019	Kudelski	Version for submission

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable Leader	Yolan Romailler (KUD)	29/10/2019
Technical Manager	Michel Abdalla (ENS)	31/10/2019
Quality Manager	Diego Esteban (ATOS)	31/10/2019
Project Coordinator	Francisco Gala (ATOS)	31/10/2019

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Description
API	Application Programming Interface
FE	Functional Encryption
FFMPEG	Fast Forward Motion Picture Experts Group (file format)
HD	High Definition
FHD	Full High Definition
IoT	Internet of Things
NALU	Network Abstraction Layer Unit
SEI	Supplemental Enhancement Information
QoS	Quality of Service
GOP	Group Of Pictures
FPS	Frames Per Second

Executive Summary

This deliverable describes the Kudelski IoT use case prototype at the end of the second year of the project. The cryptographic protocol of the use-case is described although it was not yet implemented in the demonstrator. We focused on functionality of the said prototype, and already integrated simulated dummy calls to the FENTEC library, which should allow for a smooth integration of Functional Encryption. We thus were able to successfully complete the first phase of the implementation of our prototype on time, and have a working demonstrator. This document also presents the hardware used to build the demonstrator, and the software implementation of this first version of the prototype. A section is dedicated to the performance of our prototype regarding motion detection using motion vectors, as this proved to be a significant bottleneck when implemented naively. The performance analysis allowed us to find optimizations, allowing us to significantly save bandwidth, and which should further help us enhance the encryption and decryption processes in the next phase of our prototype implementation.

1 Introduction

Following the Requirements Analysis D3.1 [1] and the Initial Technical Specification D7.1 [2] we now present the initial work into the development of the Kudelski Security IoT surveillance camera network use case prototype. Note that the Kudelski use case has been updated to a more realistic scenario but that this document retains the title of the original scenario. Throughout this deliverable the term "Local Decision Making" should be read as referring to the main subject of the work and not "Key Distribution" as in the original version before the use case was ammended. The Initial Technical Specification D7.1 [2], upon which this work is based, also refers to the ammended use case.
The goals of this document are to describe, for the Kudelski Security use case:

- how the prerequisites for prototype development have been met,
- the current state of development of the use case in terms of functionality, and
- a preliminary analysis of the performance of the prototype.

The Initial Technical Specification D7.1 [2] defined the required functionality for the prototype and how to achieve that using existing tools plus cryptographic protocols defined by the academic partners and the FENTEC library implemented by XLAB. There are also some tentative performance targets but since these proved to be difficult to quantify they are only general guidelines.
The overall goal of this phase of the project (Task 7.2) is to produce a working prototype in order to assess the suitability and efficacy of the cryptographic protocol and its ability to meet the requirements of the application. During the development of the prototype, it appeared that the computational power needed for motion detection was much greater that what the cryptographic part should require. Therefore we choose to shift the implementation of functional encryption to the next phase of the project and focus our efforts on motion detection and video handling. To this end, in this document, we concentrate on describing how our work meets the performance and functionality requirements. This will then lead into the final phase of the project where we concentrate more upon the cryptographic and security aspects of the use case while maintaining the functional requirements.
This deliverable is produced concurrently with the development of the prototypes for the other two use cases and leads into the reports on the final prototypes plus reports on performance and security:

- D7.3 First version of the truly anonymous data collection prototype (M22)
- D7.4 Final version of the truly anonymous data collection prototype (M33)
- D7.5 First version of the privacy enhanced digital currency prototype (M22)
- D7.6 Final version of the privacy enhanced digital currency prototype (M33)
- D7.8 Final version of the IoT key distribution prototype (M33)
- D7.9 First test report of the FENTEC prototypes (M23)
- D7.11 Performance report for FENTEC prototypes after first cycle (M24)

1.1 Structure of the Document

Section 2 presents the current state of the IoT use case.

Section 2.1 introduces the IoT prototype.

Section 2.2 describes the cryptographic protocol used for the prototype.

Section 2.3 gives a brief description of the platform which has been developed.

Section 2.4 outlines the structure of the software and the method of operation.

Section 2.5 summarizes what the demonstrator version of the use case application attempts to prove.

Section 2.6 explains how the prototype matches the specification and gives some brief notes on its current performance.

Section 3 concludes with a brief statement about conformance with project goals.

Section 4 gives some brief indications for the next steps in the project.

2 IoT Demonstrator

2.1 Introduction

In the modern days, with the increasing progress of computer performance, developing devices that interact with each other without requiring a human's input becomes more and more desirable. This is known as the world of IoT which currently witnesses a growing popularity. In such networks, many small devices, such as everyday objects, are interconnected in order to be controlled remotely.
Because a ciphertext usually requires to be entirely decrypted to be analysed, creating local decision making in an IoT network requires propagating a secret key over an entire chain of devices. This spreads the exposure of the key, makes the whole encryption chain susceptible to the weakest link, and can even cause privacy issues. A cryptographic primitive that solves this problem is *functional encryption* which, in addition to the common encryption and decryption capabilities, also provides the result of a function evaluated on a plaintext to certain participants. As a result, functional encryption enables making decisions depending on a function of the plaintext while keeping the ciphertext encrypted.
A typical application of functional encryption with IoT devices is a *surveillance camera network*. In this use case, a number of cameras interact with a (different) number of gateways in order to raise alarms to a host if movement is detected on the video. This scenario allows the cryptographic workload to be lifted from the cameras and the host, and distributed among the gateways. For this purpose, the video stream is fully encrypted and sent alongside a functionally encrypted stream of data corresponding to the motion on the video (namely, the *motion vectors*).
The current document describes the approach used to create a prototype of the different components of the network (i.e. the camera, the gateways, and the host). The software used to extract the motion vectors and detect movement is a modified version of FFmpeg according to the specifications of the use case described in Section 2.4. In order to emulate the components, programmable boards and a generic camera were setup as described in Section 2.3.

2.2 Cryptographic Protocol

Using functional encryption on raw video streams is a practically impossible task. Video streams are encoded using compression methods that scramble the information and, combined with the encoding itself, make it almost impossible to design an encryption scheme that would allow us to retrieve information about movement without leaking the whole video.
The IoT use case takes advantage of a workaround: embedding a list of motion vectors in the metadata of the stream. The idea is to extract the motion vectors of the video, encrypt them using some FE function, and send them along with the original video stream.
In a real world application an FE scheme would be used for motion vectors transmission, and classical end-to-end encryption for the video stream. At this phase of the prototype, no cryptographic functionality has been implemented. In the future, the IoT devices will handle two kind of encryption:

- The *Functional Encryption* of the motion vectors using a secret FE encryption key, and
- A *end-to-end* encrpytion of the video stream using non-FE encryption scheme with some symmetric secret key.

The gateway systems will not be involved in the end-to-end encryption of the video stream. They will only own an FE extraction key that allows them to perform basic motion detection without accessing the the raw motion vectors nor the video stream. The backend system will share the secret encryption key with the IoT devices and thus be able to decrypt the video stream.

The protocol describing the different interactions between an IoT device and a gateway is shown in Figure 1.

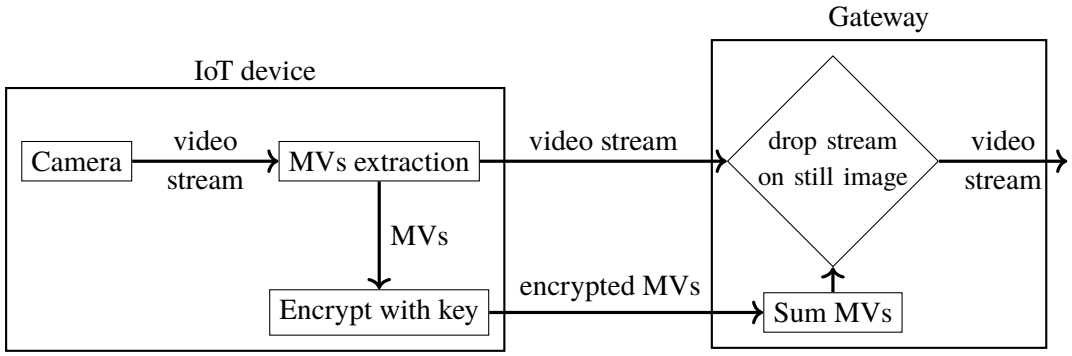

Figure 1: Protocol among the components of the FENTEC surveillance camera network prototype.

2.3 Platform

The three different components involved in the use case were emulated as follows:

- The *camera* is a Logitech C922 PRO.
- The *encryptor* is a Raspberry Pi 4 Model B 2GB RAM.
- The *local decision maker* is a Raspberry Pi 4 Model B 2GB RAM.
- The *decryptor* is a general-purpose laptop.

All the above items were interconnected according to Figure 2. All devices must run the Fentec specific version of FFmpeg.

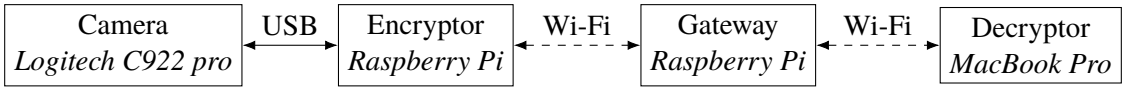

Figure 2: Setup for the IoT surveillance camera use case prototype.

Camera	Logitech C922 PRO. 1920x1080 pixels at 30fps in mjpeg format
Encryptor	Raspberry Pi 4 Model B 2GB RAM. Dummy encryption key, amount of motion vectors to carry.
Local decision maker	Raspberry Pi 4 Model B 2GB RAM. Motion detection threshold.
Decryptor	MacBook Pro No parameters.

Table 1: Hardware components and their parameters

2.4 Software

This section details the software choices made to fulfil the use case requirements.
Requirements

1. Extract data related to motion on a provided video stream.
2. Encrypt the motion-related data with functional encryption in such a way that a motion detection function can be queried on the ciphertext.
3. Encrypt a stream of video with symmetric encryption.

Overview
To fulfil all of the above criteria, the software for the prototype was based on *FFmpeg*, which is a vast free and open-source software suite to handle multimedia content. It offers the possibility to modify video and audio streams by adjusting default parameters or by applying filters, which can be either provided by FFmpeg or implemented by the user. Therefore, FFmpeg is useful to transcode the camera video (i.e. set the desired codec, GOP, frame-rate, etc.) as well as for using the custom filters.

2.4.1 FFmpeg

FFmpeg helps us to manipulate motion vectors contained in H.264 video stream for encoding purposes. The idea is to correlate the low-level displacements of pixels described by motion vectors with the high-level interpretation humans have of motion. This will be leveraged using custom filters implementing functional encryption of extracted motion vectors and full encryption of video stream.
A *motion vector* represents the displacement between two frames of a macroblock, which is a continuous block of pixels in a frame. Such a vector is an essential aspect of video compression, as consecutive frames can now be drawn predictively, i.e., according to a reference frame instead of permanently encoding the whole frames.
Modifications summary

- Captured video is a H.264 FHD 30 FPS transcoded video stream with GOP 30.
- A custom bitstream filter `h264_fe_mv_crypt_bsf` which encrypts the motion vectors with functional encryption and embeds them in the video stream, and encrypts the video stream at NALU level using symmetric encryption.
- A custom bitstream filter `h264_fe_mv_extract_bsf` which extracts the encrypted motion vectors and detects movement according to the functional encryption principles and transmits the frames to the backend if motion is detected.
- A custom bitstream filter `h264_fe_mv_decrypt_bsf` which decrypts the received frames.

Note that for now, although the video stream already passes through the different filters, no cryptographic function is implemented.

Motion vector data structure
Once extracted the motion vectors go through a basic processing to compute their norms weighted by the surface of the corresponding macroblock. Then, they are filtered so that we only keep the relevant ones and carried in small fixed sized SEI NALU, which are interleaved with frame NALU in the video stream.

2.4.2 H.264

The H.264 codec converts **frames** into three main types :

I-frames also known as intra-coded frames, they have no reference and describe an entire image.

P-frames also known as predicted frames, are usually described with respect to the previous I-frame or P-frame by the displacement of macroblocks following motion vectors, which is, in general, more efficient than encoding the entire image. This requires the reference frame to be decoded before.

B-frames also known as bi-directional predicted frames, similarly to P-frames, are usually described with respect to the closest previous and next I-frame or P-Frame. This requires the reference frames to be decoded before even if they are displayed after in time.

The **GOP** acronym stands for *Group of pictures*. It defines the amount and type of frames between two full images (I-frames). A GOP of 30 associated with 30 fps allows us to have one full image every second. The backend laptop is thus able to read a proper video stream at most one second after movement occurs.

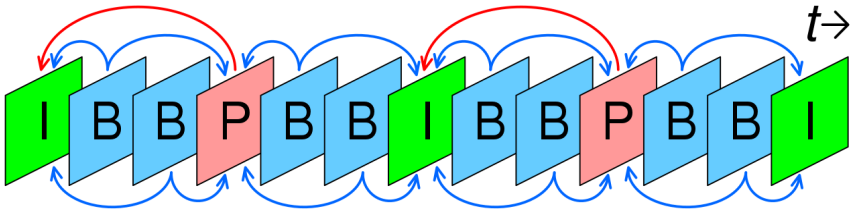

Figure 3: Group of pictures

A typical value for our videos' GOP is :

`IBBBPBBBPBBBPBPBBBPBBBPBBBPBBP`

We choose the previous parameters so that we get a reasonable frame rate as well as exactly one I-frame per second. This allows us to decide whether to keep or to cut the video stream second by second and keep it decodable.

2.4.3 Procedure

Camera
The camera produces a video stream encoded in mjpeg, the native picture size and framerate already correspond to the requirements (1080p at 30 fps). The IoT device still has to transcode that stream into a h264 stream, drop the audio part and enforce a GOP of 30. The `ultrafast` preset ensures that the Raspberry Pi is able to transcode the stream in real time. For this purpose, the IoT device runs FFmpeg with the following parameters :

```
ffmpeg -f v4l2 -framerate 30 -video_size 1920x1080 -c:v mjpeg -i input -an -c:v
                        h264 -preset ultrafast -g 30
```

Encryptor
The encryptor is in charge of extracting the motion vectors and inserting them in the video stream, using the `export_mvs` flag. Instead of carrying all the information of the motion vectors, we only keep their norms weighted by the surface of the corresponding macroblock. Until now, only P-frames' motion vectors have been considered because they seem to be more consistent from frame to frame. The encryption will be done using the `h264_fe_mv_crypt_bsf` bit stream filter which applies functional encryption on motion vectors and packs them into SEI NALUs. For now, the filter packs motion vectors in plaintext into SEI ALUs. The IoT device runs a second instance of FFmpeg dedicated to the cryptographic part :

```
ffmpeg -y -flags2 +export_mvs -i input -c:v copy -bsf:v
                'h264_fe_mv_crypt=key=key'
```

Gateway
For each frame NALU, the gateway receives a SEI NALU (containing the motion vector norms of the corresponding frame) and forwards the frame NALU only when Motion = TRUE according to Figure 4. The decryption and decision making is handled by the `h264_fe_mv_extract` bit stream filter, which takes a decryption key and a threshold number as parameters.

Backend
If motion is detected, the backend device receives the video stream (stripped of motion vectors) and displays it using an FFmpeg video player. When no more data is transmitted the backend continuously displays the last decoded frame. As the video stream was not encrypted, there is no need to use the `h264_fe_mv_decrypt` bit stream filter.

2.5 Demonstrator

The goal of the demonstrator is to show that functional encryption can be applied on video streams on the fly by small embedded devices (namely, Raspberry Pi micro-computers) and that it enables us to use less bandwidth between gateways and backend systems in such a configuration than in a traditional video surveillance infrastructure.
The cryptographic, video transcoding, and motion detection algorithms should be able to run in near real time on small CPUs (ARM 4 cores at 1.5 GHz), and motion detection should be accurate enough to practically detect obvious movements, like a single person walking in front of the camera.

The demonstrator for the Kudelski IoT use case uses the modified version of FFmpeg, installed on each device. The FFmpeg framework is called by successive shell scripts :

- `backend.sh` runs on the backend laptop. Listens on port 2000 for any video stream and plays it directly.
- `gateway.sh` runs on the gateway Raspberry Pi, listens on port 1500 for any video stream, apply the `h264_fe_mv_extract` bit stream filter and forwards the output to the backend.
- `camera.sh` runs on the encryptor device and handles the crypto part of the IoT device. It listens on port 12345 for a video stream and apply FENTEC encryption on it. The output is forwarded to the gateway.
- `webcam.sh` also runs on the encryptor and communicates with the camera device. It specifies what format should the camera outputs, and transcode it to an h264 stream. It forwards the output on port 12345 locally.

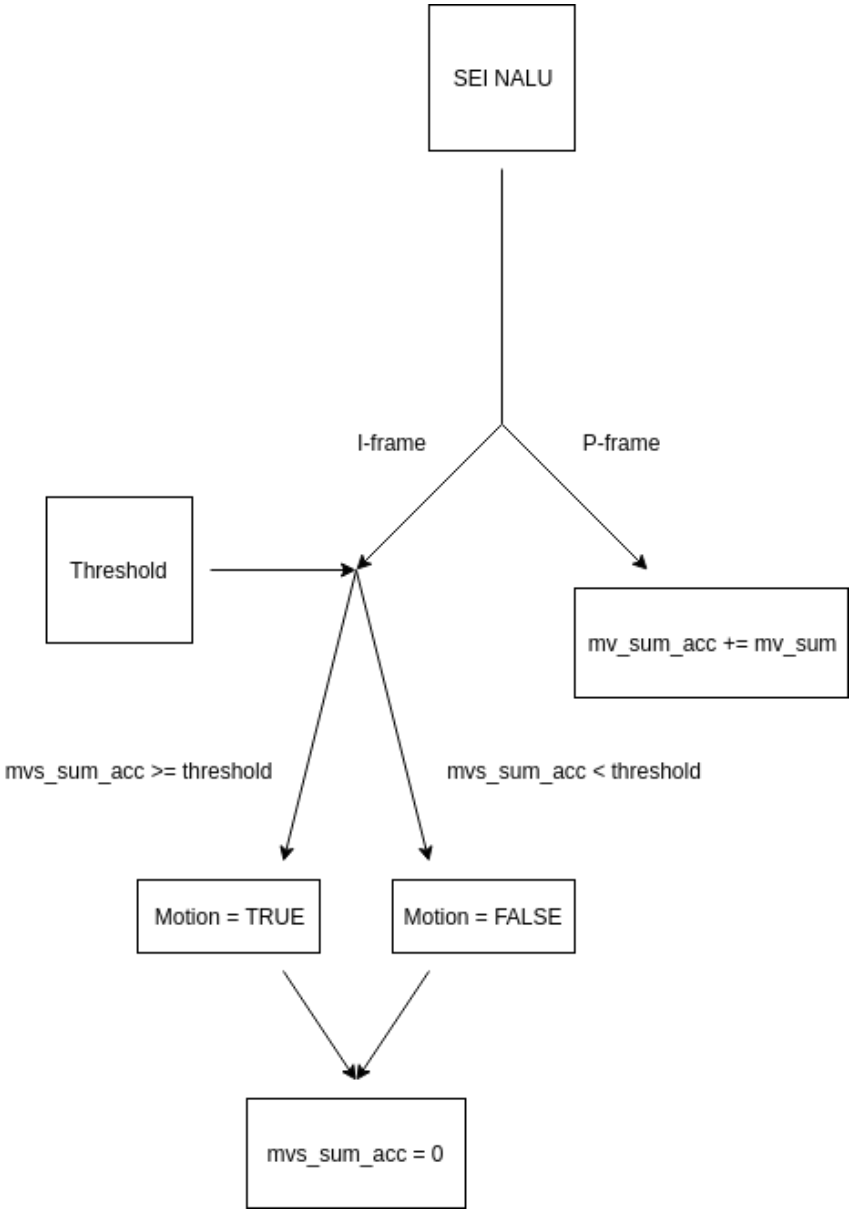

Figure 4: Motion Detection Flow Chart

2.6 Performance and Security

One of the goals of the IoT use case prototype is to reduce the bandwidth usage between the backend device and the decision-making gateway. The main challenge posed by the current design presented in Figure 1 is the bandwidth usage overhead between the IoT device and the gateway caused the presence of the encrypted MVs. As motion vectors have to be in cleartext before being encrypted, to be able to leverage the functional encryption scheme correctly, there is no possibility to compress them before the encryption step. But as the output of the encryption is hardly distinguishable from random noise, attempting to apply a compression algorithm on them after the encryption would not reduce the amount of transmitted data.

2.6.1 Benchmarking dataset

We used 4 videos for this project. They all simulate video surveillance streams collected from car parks. "Video a cropped" is the same as "video a" except that we only kept the right bottom corner, where most motion happens.

Figure 5: Video a - First Frame

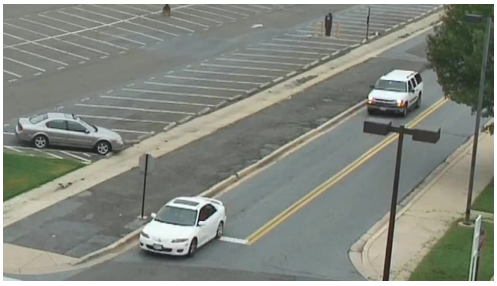

Figure 6: Video a cropped - First Frame

Figure 7: Video b - First Frame

Figure 8: Video c - First Frame

Encoding
All the videos have been transcoded with the following properties :

- FHD H.264 video stream
- no audio stream
- 30 FPS
- GOP 30
- 2000 $\pm$ 100 kb/s bitrate

This can be reproduced with any other video using the following FFmpeg command :

```
./ffmpeg -i in -c:v libx264 -an -s 1920x1080 -r 30 -g 30 -b:v 2000k -bt 100k out
```

2.6.2 Motion detection

The first implementation used a constant threshold parameter and allowed us to distinguish moving frames from still frames. Nevertheless, it is hard to find the ideal threshold, which would detect smaller motions like a person walking when the video stream is particularly noisy (i.e. strong wind and moving tree leaves) and is also showing big vehicles moving. It must be mentioned that this implementation transmits or drops frames according to the state computed from the previous GOP so it may suffer from a one second delay in the motion detection.

Histograms of the sums of motion vector norms

Here are the motion vector norms sums per second histograms for our dataset. The threshold (in blue) is set by hand analysing motion in the video. Red bars above the threshold trigger the motion detection while green ones do not.

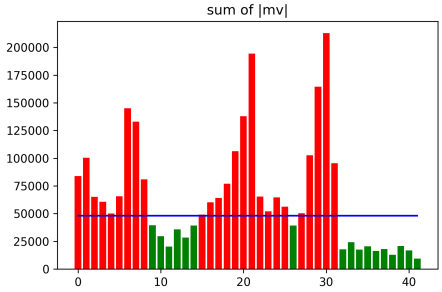

Figure 9: Video a - Motion Vector Norms Sums

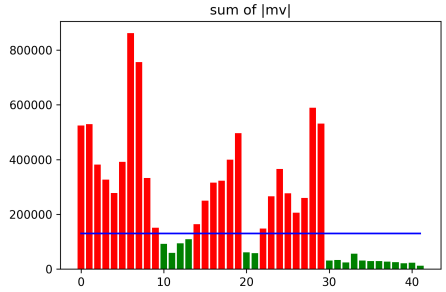

Figure 10: Video a cropped - Motion Vector Norms Sums

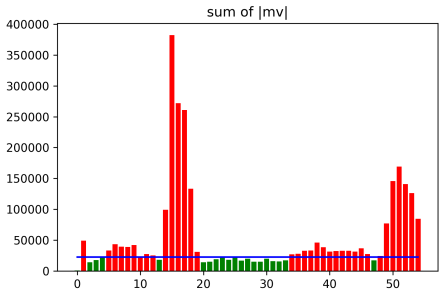

Figure 11: Video b - Motion Vector Norms Sums

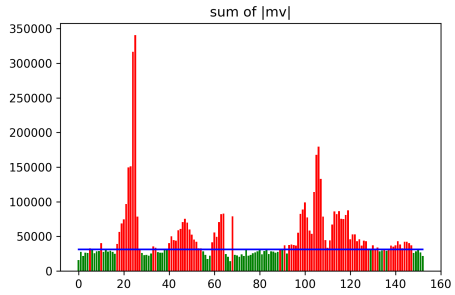

Figure 12: Video c - Motion Vector Norms Sums

Issue

The main point shown in Table 2 and Table 3 is that the carried fixed length SEI NALU requires a lot of bandwidth between the camera and the gateway. As expected, we observe an overall gain of bandwidth even if it essentially happens between the gateway and the backend.

Video	Duration	Initial File Size	Camera-Gateway Data	Gateway-Backend Data
a	41 s	11 MB	98 MB	7 MB
a cropped	41 s	11 MB	98 MB	7 MB
b	55 s	15 MB	127 MB	9 MB
c	153 s	40 MB	354 MB	24 MB

Table 2: Current bandwidth usage

Video	Camera-Gateway Overload	Camera-Backend Gain	Discards
a	+ 790 %	- 20 %	40 %
a cropped	+ 790 %	- 20 %	40 %
b	+ 750 %	- 40 %	35 %
c	+ 785 %	- 40 %	40 %

Table 3: Current impacts

2.6.3 Include SEI NALU only for I-frames and P-frames

The first step of optimization is to drop unexploited SEI NALU since motion detection seems to work reasonably well without considering B-frame motion vectors. Adversaries should not be able to take advantage by deducing the GOP. I-frames do not have associated motion vectors but we still need to include SEI NALU in order to identify them. If this technique is viable then further changes can be made so that I-frames SEI NALU only carry their type and no extra space for their inexistent motion vectors.

Results
The bandwidth usage is better than with our current implementation even it is still not satisfying to use three time the bandwidth according to Table 4 compared to simply stream the video.

Video	Camera-Gateway Overload
a	+ 230 %
a cropped	+ 230 %
b	+ 230 %
c	+ 240 %

Table 4: Only I-frames and P-frames motion vector norms

This uses exactly the same references as our current implementation, then the motion detection performs exactly the same.

2.6.4 Smaller fixed size SEI NALU

This option profits from the fact that most of the motion vectors have negligible impact for the final motion detection. Assuming this, we can choose a smaller fixed size data structure to carry filtered motion vectors, whose norm is equal to zero to save bandwidth between the camera and the gateway. This is an extension to the previous improvement. Using fixed size SEI NALU, this implies that due to variance and outliers that we will have to discard motion vector norms even if they are non-zero. Choosing which motion vectors we keep also becomes a challenge in order not to take a biased sample of them.

Results
According to Table 5, most of the motion vectors are irrelevant. Nevertheless, we should notice that this depends upon the type of video. Our videos a, b and c capture wide and far fields, so, motion vectors are usually small. On the other hand, video a cropped focuses on the motion region, as a consequence, motion vectors have bigger norms and happen more often. This would be a biased scenario assuming that most of surveillance cameras capture the wider field possible.

Video	Minimum proportion of zero |MV|	Median proportion of zero |MV|
a	74 %	91 %
a cropped	36 %	82 %
b	51 %	94 %
c	38 %	91 %

Table 5: Proportions of zero motion vector norms

The following plots show the proportion of zero and non-zero motion vector norms for all the P-frames of each video. The red line corresponds to the minimum proportion of zero motion vector norms and the green one refers to the median proportion of zero motion vector norms.

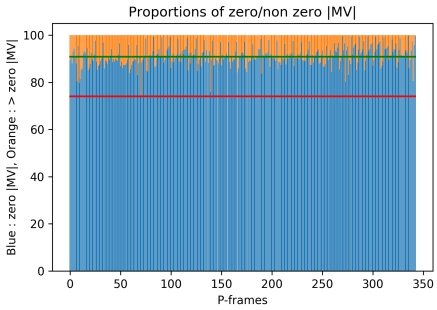

Figure 13: Video a - Motion Vectors Proportions

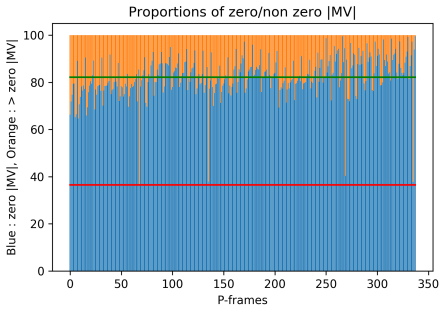

Figure 14: Video a cropped - Motion Vectors Proportions

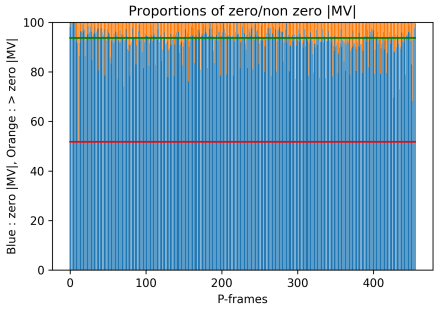

Figure 15: Video b - Motion Vectors Proportions

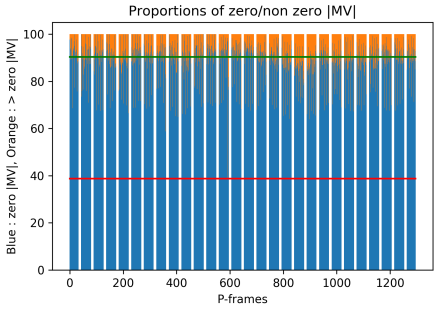

Figure 16: Video c - Motion Vectors Proportions

For this technique, we carry 2% of the maximum number of motion vectors (and not the effective number of motion vectors available in each frame), which is close to the median proportion of zero motion vectors for our videos. Notice that this parameter should be tuned specifically for each video and cannot be predicted. The motion vector norms array is sorted in decreasing order and we keep the largest values. This results in a noticeable improvement as shown in Table 6.

Video	Camera-Gateway Overload
a	+ 10 %
a cropped	+ 20 %
b	+ 30 %
c	+ 5 %

Table 6: Fixed size SEI NALU

These numbers depend on how many motion vectors were initially available in the video and the maximum number of motion vectors.
As we may discard motion vector values, then the threshold may need to be downscaled. Even if we discard some motion vector norms, we are able to achieve similar results compared to other techniques. This means accuracy of the detection is not an issue.

2.6.5 Variable size SEI NALU

Finally, we evaluate the most flexible technique: carrying all the non-zero motion vector norms. By doing this, all the information we need to perform motion detection is available and we reduce the bandwidth usage by the greatest amount. We will pay special attention to see if the size of the SEI NALU is related to the amount of detected motion. The advantage of using variable size SEI NALU is that we do not lose any information and we do not carry any useless information.

Results
As shown in Table 7, using variable size SEI NALU leads to a bandwidth overhead of at most 20% on the benchmarking dataset. This is how the current version of the prototype is implemented.

Video	Camera-Gateway Overload
a	+ 10 %
a cropped	+ 20 %
b	+ 2 %
c	+ 6 %

Table 7: Variable size SEI NALU

2.6.6 Issue

As we are only keeping the non-zero motion vector norms, the SEI NALU of these may be correlated to their sum. This is undesirable since any adversary could deduce the presence of motion in the video stream just by looking at the SEI NALU size. The following tables show the sum of motion vector norms and the amount of motion vector norms per GOP. We observe a slight tendency the sum to be greater when the SEI NALU are bigger. The threshold is drawn in blue on the plots. An adversary could, for each video, choose a threshold on the SEI size that would directly correlate to the motion vectors number and be able to

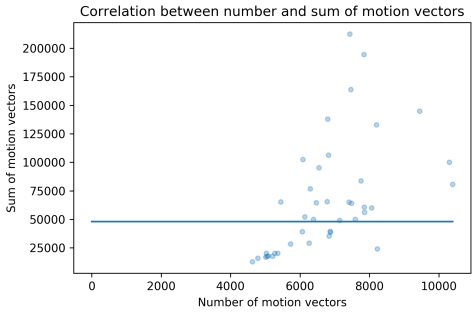

Figure 17: Video a - Correlation

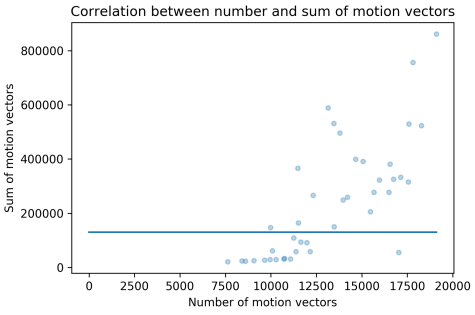

Figure 18: Video a cropped - Correlation

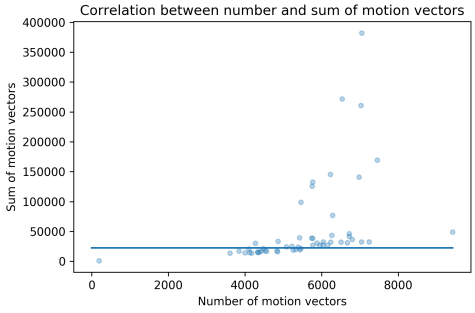

Figure 19: Video b - Correlation

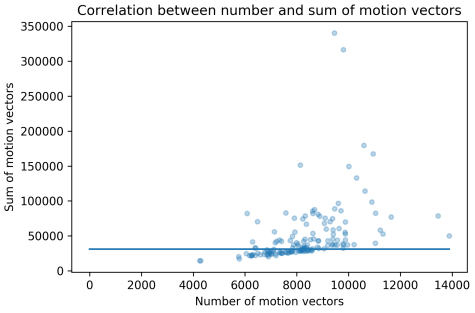

Figure 20: Video c - Correlation

determine quite accurately when movement occurs. The idea of variable size SEI NALU should therefore be rejected.

3 Conclusion

In this document, we presented the first version of Kudelski's IoT Video Surveillance use case. For this initial version, we concentrated upon functionality and solving the problems arising from our detection method, namely the large bandwidth increase caused by the presence of the encrypted motion vectors embedded in the video stream.
We have a demonstrator for the prototype which can be used to show how motion detection can be performed using the sum of the motion vector, which can be computed using an inner product functional encryption scheme evaluated at the gateway level.
We are currently able to demonstrate the full pipeline of our prototype, from a camera to a backend system, and to perform the required local decision making at the gateway level, which enables us to save bandwidth between our gateway and our backend system as we wanted.
We also addressed the performance problems we faced, discussed extensively the bandwidth usage issue of our current implementation of the FENTEC prototype and proposed multiple methods to reduce the bandwidth usage. It appears that dropping the B-frames SEI NALU provides a good improvement with limited side effects. Then, we figured out that most of the motion vectors norms in the video stream are useless since their norms are zero and therefore do not change the motion detection results. Since the variable size SEI NALU leak information, the best trade-off is to set smaller fixed size SEI NALU. Even if they they may discard non-zero motion vectors norms, they are almost as precise as the original technique by adjusting the gateway threshold.
We also want to stress here that reducing the number of motion vectors that needs to be encrypted should dramatically increase the encryption performance, which we plan to address in the next phase of our prototype development.
In summary, the prototype is at the stage expected for this phase of the FENTEC project. We have a working prototype with which we can move forward and develop to use functional encryption in the later stages of the project.

4 Next steps

Now that motion detection works on the IoT suveillance camera prototype, we will focus our work on the cryptographic part of the prototype. The next step will be to implement real functional encryption on motion vectors. The motion detection will have to be slightly adapted, as the current FE algorithms does not permit simply summing the norm of encrypted vectors. Once FE is fully implemented we will be able to apply end-to-end encryption on the video stream, thus enforcing security on the prototype.

References

[1] FENTEC. D3.1 technical requirement report analysis. Technical report, European Commission, 2018.

[2] FENTEC. D7.1 preliminary specification of fentec prototypes. Technical report, European Commission, 2018.