Advanced multimedia applications

Kiani Lannoye
Tim Deweert
Robin Praet
Jan Van de Vyver

Video chat application - data hiding

2014-2015

Contents

1 General information

The goal is to create a video chat application. To prevent eavesdroppers from listening in on private conversations, the real conversation could be hidden in a video. This message can be directly embedded in the uncompressed video. However, since this video needs to be encoded before sending it over the network, part of the message might be lost due to compression artefacts. A more robust approach could be to apply a data hiding algorithm directly to a compressed bit stream. The decoder at the receiver-side can then correctly extract this message if the key is known. A comparison is made between hiding the data before or after video compression.

2 General project decisions

2.1 Decisions

The period before every meeting with the teaching staff is considered as an iteration.

First iteration (Course introduction - 26/02/2015)

- Git with GitHub will be used as distributed versioning control system for the code. (13/02)
- Jan is responsible for the progress report throughout the project (i.e. make sure it is uploaded in time and correct). (18/02)
- Jan is in charge of the communication with the external parties. (18/02)
- For scalability purposes, a web application is chosen over a native application. (18/02)
- A frequently mentioned library for video chat applications in a web browser is WebRTC. Question remains whether or not the sent data can be modified. (21/02)

Second iteration (27/02/2015 - 12/03/2015)

- The OpenCV library will be used for capturing the webcam data WebRTC library. The reason for the use of OpenCV is that it is hard to intercept the data captured by WebRTC and adapting it before sending it over a network, while OpenCV provides easier access to this data (in order for us to apply steganography). (4/03)
- A Windows application is chosen instead of a web application. The reason for this is that OpenCV provides better integration with the C++ language and the fact that creating a GUI for this application will be simplified, so we can have an increased focus on the data hiding aspect of the project. Another possibility was to use Python together with OpenCV, but we would need to have an intermediate webserver for the conjunction with the GUI and the data hiding algorithms, which interferes with our P2P purpose. (4/03)

Third iteration (13/03/2015 - 26/03/2015)

No decisions have been added in this iteration.

Fourth iteration (27/03/2015 - 30/04/2015)

In a meeting with the client, some changes to the project scope have been made. We noticed that when encoding using HEVC, even with the lightest compression settings, it is still hard to decode text from frames we encoded text into. The client is aware of the fact that the encoding is one of the most severe attacks on

the entropy of the data, finding an algorithm that is completely robust against the encoding would lead us too far. As the project will be used for demo-using purposes, the main feature is the architecture and how extendable the software is. Also, encoding text into the output data *after* encoding is even harder, which would again lead us too far. We should be able to show two scenarios at the end of the project:

- Encoding text into a YUV video and retrieving the text from this video (without encoding) over a network
- Encoding text into a YUV video, encoding this video and retrieving the text from the decoded video over a network. This can be shown with different levels of compression when encoding, to show how robust the encoded text is.

Fifth iteration (01/05/2015 - 12/05/2015)
We had some extra meetings with our client about the decoder. There were some problems with including and building the libraries of ffmpeg but eventually we got the decoder working. We also improved our steganography algorithm and the sending and receiving with TCP. And because one of our teammates didn't complete his task of making a GUI, we had to implement a commandline interface as GUI.

2.2 Responsibilities

First iteration (Course introduction - 26/02/2015)
No clear responsibilities have been given here, every team member should investigate what libraries we can use to stream video data from one client to another.

Second iteration (27/02/2015 - 12/03/2015)
As we have thought through the architecture and what libraries to use, we have now divided the team into front end and back end parts.
Front end: Robin Praet
Back end: Kiani Lannoye, Tim Deweert, Jan Van de Vyver

Third iteration (13/03/2015 - 26/03/2015)
Same responsibilities as in the second iteration.

Fourth iteration (27/03/2015 - 30/04/2015)
This is an integration iteration, so everyone is responsible to integrate their own components. Tim Deweert: Getting the x265 encoder to work with our program. Robin Praet: Integrating the GUI into our program and converting TCP to UDP. Kiani Lannoye & Jan Van de Vyver: Applying steganography to the webcam data.

Fifth iteration (01/05/2015 - 12/05/2015)
This is our last iteration so we had to integrate everything. Tim Deweert: Getting the ffmpeg decoder to decode the hevc stream and integrate it in our project. Robin Praet: Integrating the GUI into our program. Kiani Lannoye & Jan Van de Vyver: Improve sending/receiving and steganography. Kiani Lannoye: write a commandline interface as GUI.

2.3 Requirements

Must-haves

- Video chat with a specific user
- Support for raw data hiding
- Support for HEVC coding and HEVC data hiding

Nice-to-haves

- Database to keep track of contacts

Quality attributes

- Modifiability: The application should be easily extendable (easy implementation of support for new video codecs)
- Security: Obviously, as we are working with steganography
- Performance: The application should have a high performance in terms of applying steganography and encoding-decoding the video as we work in C++ instead of JavaScript.

2.4 Risk list

First iteration (Course introduction - 26/02/2015)

- WebRTC seems like a solid solution for video conferencing, but we are not sure whether or not we can apply steganography to the sent data.

Second iteration (27/02/2015 - 12/03/2015)

- Finding a suitable encoder and decoder for the HEVC standard in C/C++.
- Feeding consequent images to the encoder as an actual video stream.

Third iteration (13/03/2015 - 26/03/2015)

No new risks have been added in this iteration.

Fourth iteration (27/03/2015 - 30/04/2015)

- Not being able to integrate the FFMPEG decoder into our project in time.
- Finding a good way to store the text in the video to make it more robust against compression algorithms of HEVC and others.

Fifth iteration (01/05/2015 - 12/05/2015)

Not getting the GUI on time and not sending the data via UDP.

2.5 APIs & Frameworks

First iteration (Course introduction - 26/02/2015)

- WebRTC is an option for the video capturing and the P2P transmission, question remains whether or not we can actually apply steganography on the data.
- Peer.js is a library on top of WebRTC that wraps the browser's WebRTC implementation to provide an easy-to-use P2P connection API.

Second iteration (27/02/2015 - 12/03/2015)

- OpenCV is a library available for C++ and Python that has a strong focus on real-time applications with the provision of several algorithms and solutions for these applications.

- Winsockets is a library that comes with Visual Studio C++, and we are using the library to make sockets to work together with ZeroMQ to be able to make a TCP connection between two instances of the application to setup a P2P connection.
- ZeroMQ is a library used to bind sockets for a TCP connection.

Third iteration (13/03/2015 - 26/03/2015)

No new frameworks or APIs have been added in this iteration.

Fourth iteration (27/03/2015 - 30/04/2015)

- FFMPEG will be used as a decoder for the HEVC standard.

Fifth iteration (01/05/2015 - 12/05/2015)

No new frameworks or APIs have been added in this iteration.

3 Prototype

First iteration (Course introduction - 26/02/2015)
The first prototype is a simple P2P video conference application where the user ID is generated by the server and this ID should be passed in another way than the application itself. It's written in HTML5 combined with the Peer.js library discussed in the APIs & Frameworks section, and looks like the screenshot on Figure 1. Calls are automatically answered and no concurrency checks are built in yet.

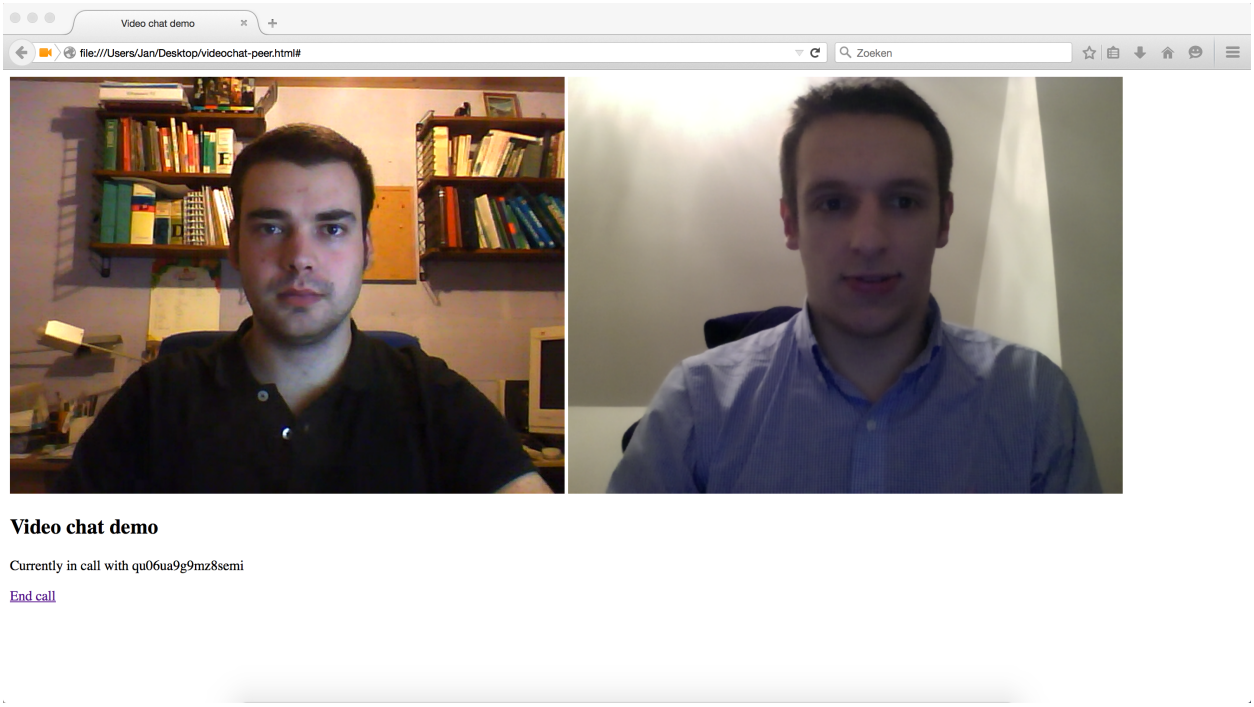

Figure 1: Initial prototype example

Second iteration (27/02/2015 - 12/03/2015)
The second prototype no longer uses Javascript or WebRTC to capture or send data. For now we use C++ code with the OpenCv library. We have a sending and receiving side. We capture images from the webcam and send them over the network to a specified IP address using the TCP protocol. At the receiving side we display the images one by one. Also, we have come up with an architecture for the application, based on the components we think are required. The architecture is given on the next page.

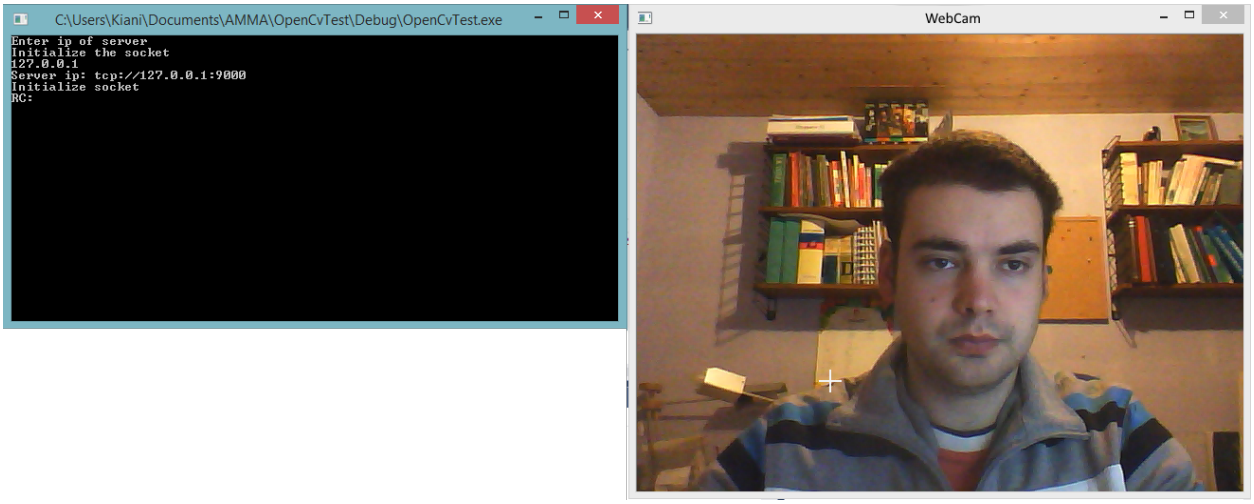

Figure 2: Second prototype example

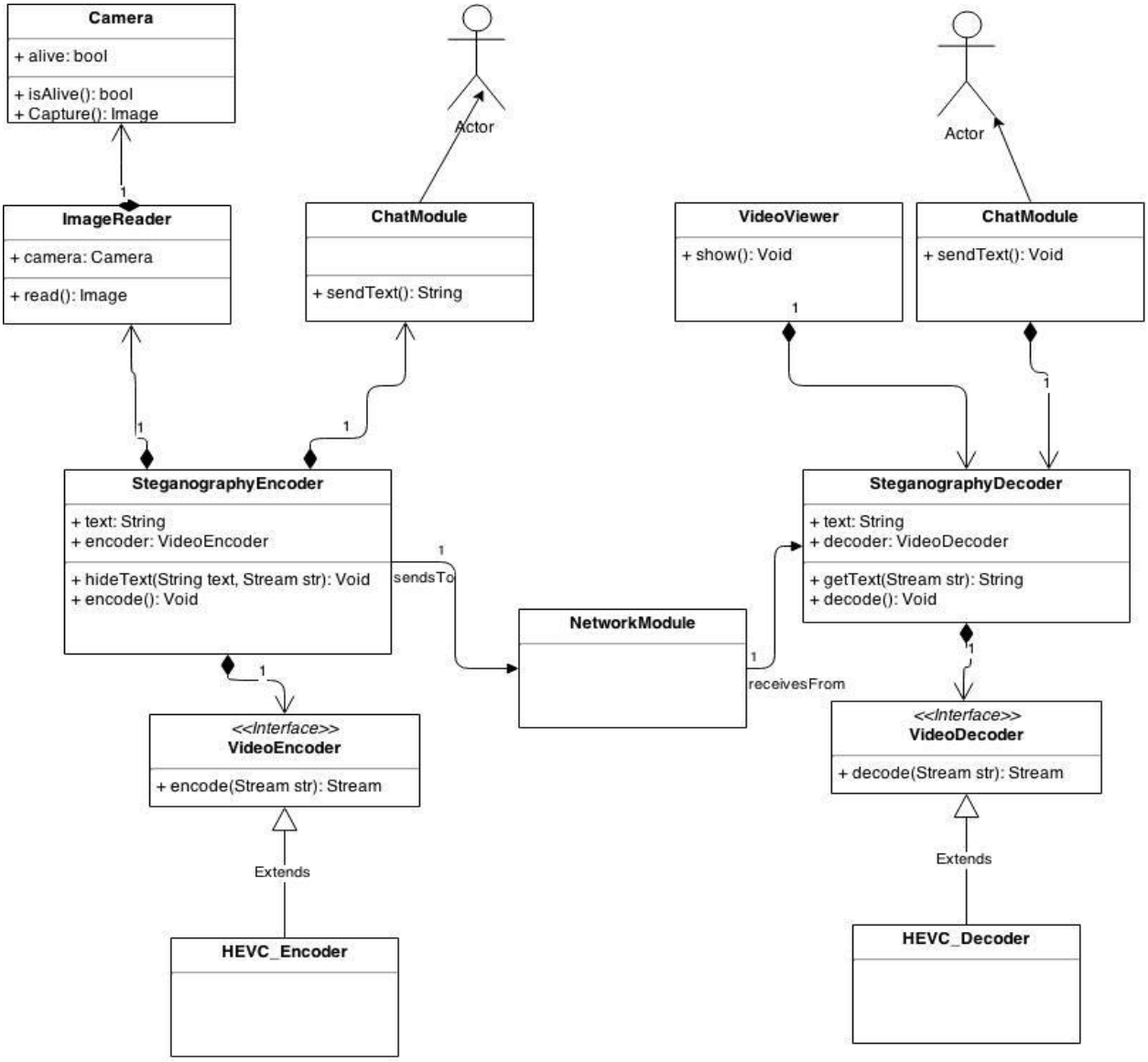

Figure 3: Initial architecture

Third iteration (13/03/2015 - 26/03/2015)
The third prototype did not improve nor focus on the sending or retrieving of the video, but we focussed on the embedding of text into the video. The retrieving of the video and sending and receiving of the videoframes, still remains the same as in the previous prototype. In the picture below we see the original image, with no text embedded yet. Then we execute the test program to apply steganography. We succesfully retrieve the text from the adapted image, which is indeed the same as the text that was embedded. For this example we use the most significant bits (MSB) to embed the text, to show clearly the degradation of the image after applying steganography. When the steganography is finished, we have an altered image, for which we clearly see the top left pixels have changed. These pixels hold the information about the embedded text.

Fourth iteration (27/03/2015 - 30/04/2015)
The GUI is now finished.

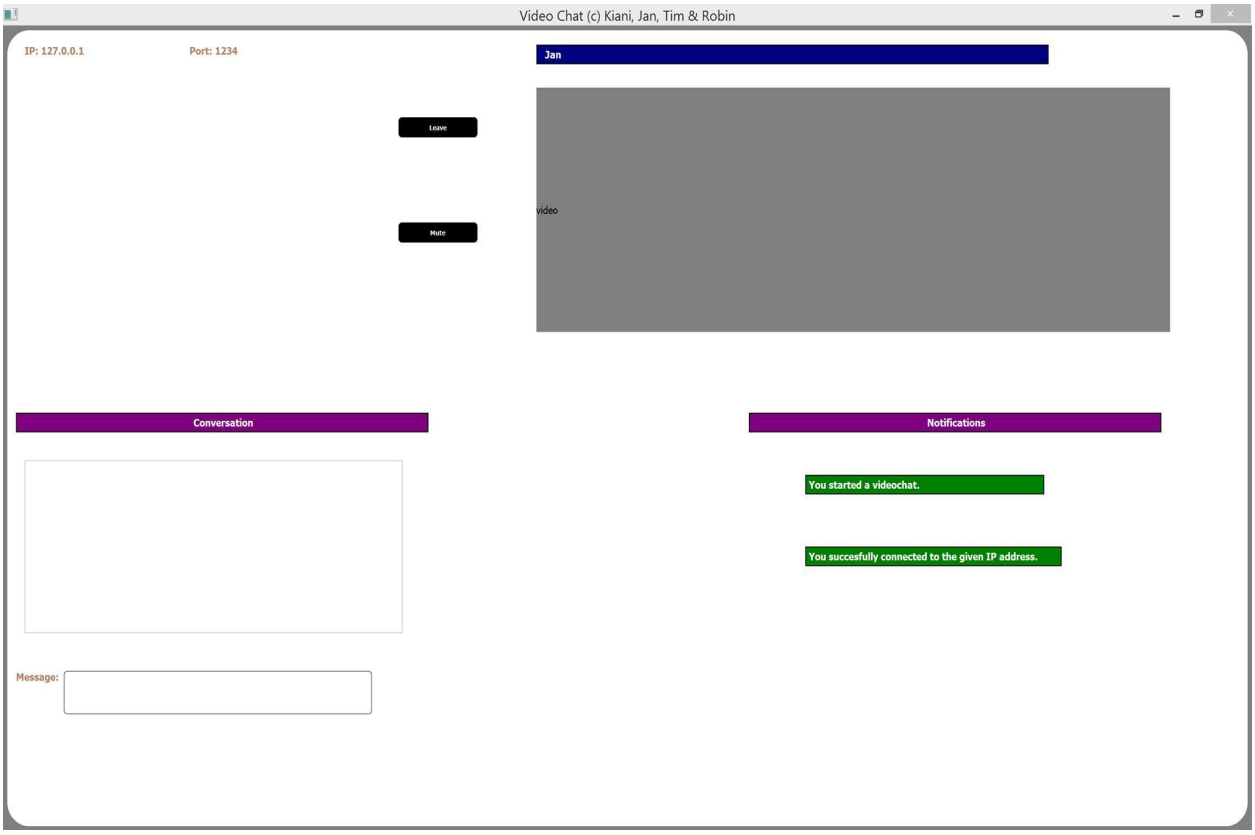

Figure 4: GUI

The encoder is now successfully integrated into our program. We can now encode text directly into the YUV data from the webcam and send this raw YUV data or first encode it with x265, which gives following result.

Figure 5: YUV webcam data after encoding and decoding

We can apply steganography to the raw YUV data and then retrieve this encrypted text. For this example, we put the text "Dit is een test" into every frame and decoded every frame.

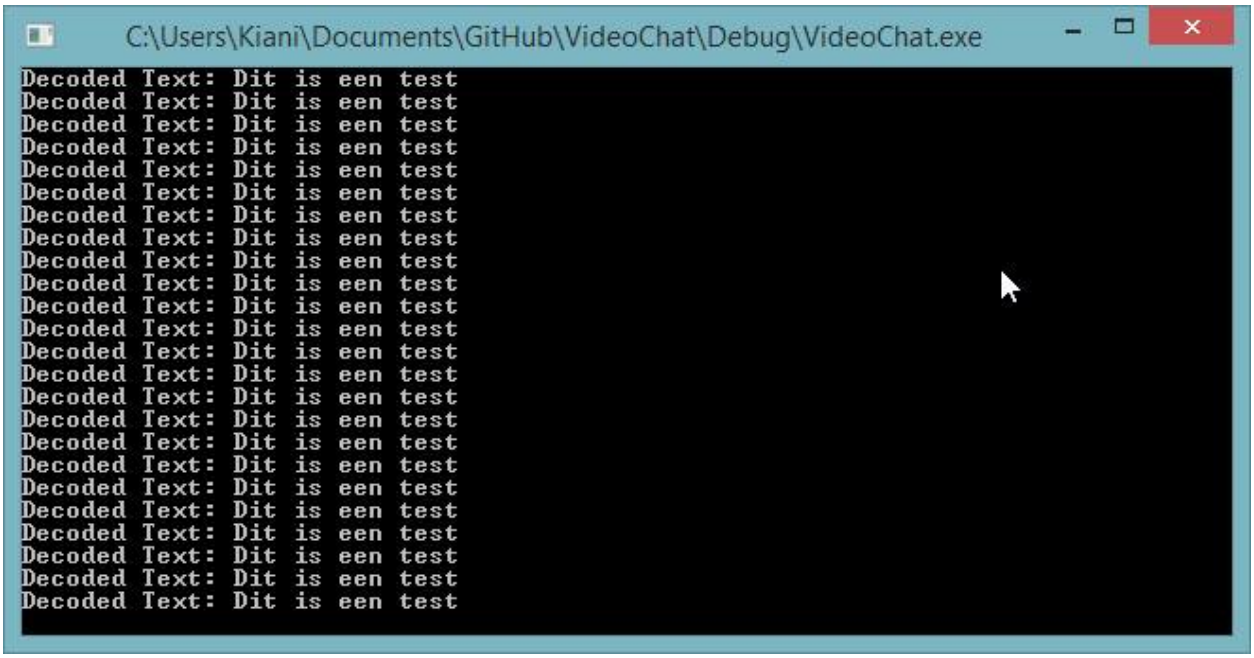

Figure 6: Steganography text retrieval in raw YUV data

We can also encode the video after applying steganography and then (manually) decode the output and try to retrieve the text from there. Depending on the different encoder settings, we are able to retrieve parts of the text (due to the quality loss) and at minimal settings even the complete text. Due to not being able to decode on the fly yet, we tried to retrieve the text of one frame. As you can see the text is only partially retrieved. We are now planning to search for better parameters to make the text encryption more robust and still enjoy a lagfree video chat framework.

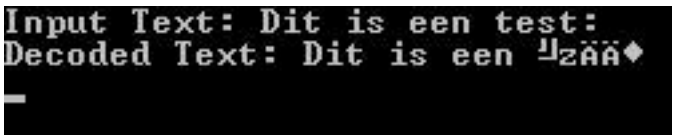

Figure 7: Steganography text retrieval after encoding

Fifth iteration (01/05/2015 - 12/05/2015)

We can now encode and decode the video in HEVC format. As in previous iteration, Before encoding the YUV, we can apply steganography to the raw YUV data and after then after decoding retrieve this encrypted text.

Figure 8: Steganography text retrieval after encoding and decoding

The retrieval of the encrypted text greatly depends on the encoder settings.

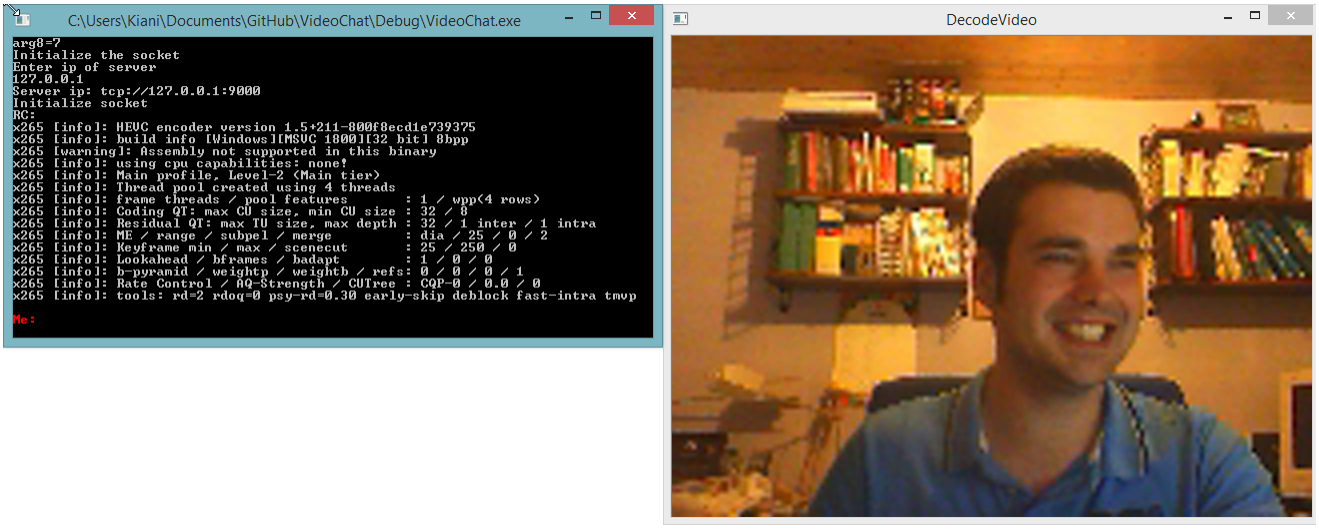

Figure 9: YUV webcam data after encoding and decoding

4 Application

The application is started through the command line with the following options:

- Required: To encode the stream or just send raw data (true or false)
- Optional: resize the video at the receiver end to show a larger video (true or false)

- Optional: lookahead buffers (HEVC specific integer)
- Optional: qp (quantization parameter, 0 = no quantization)
- Optional: bframes (HEVC sepcific integer for max number of b-frames)
- Optional: keyint (HEVC specific integer to indicate Max intra period in frames)
- Optional: minkeyint (HEVC specific integer to indicate Minimum GOP size)
- Optional: Bit to use to store string (0-7: 0 = LSB, 7 = MSB)

Next up you need to enter the ip address of your partner who is running the same application. Once your partner enters your ip, you will see his video stream. You can now start typing your messages into the command line. Your messages will be colored red, the messages from your partner will be colored light-blue. Once you hit esc (when the video is focussed as a window) the application will exit. You can also type "stop" to achieve this functionality.

As mentioned before, the GUI is a command line based GUI, because a group member had some personal problems and couldn't finish the integration.

5 Overview of the different meetings

5.1 Initial meeting with client - requirements (18/02/2015)

- **Present:** Everyone (Kiani Lannoye, Tim Deweert, Robin Praet, Jan Van de Vyver)
- **Goal:** Getting to know the specific details and requirements (if any) for the project.
- **Meeting notes:** We had some questions about what frameworks or languages were expected to be used, how scalable the application needed to be (e.g. use of logins or not), whether or not network errors should be taken into account. General reply of the client is freedom. We are free to use whatever framework we want, whatever language we want, as long as the focus is on the aspect of steganography and modifiability. The application should be easily extendable, meaning support for a different video encoder/decoder should be easily added to the application. So the focus lies more on architectural robustness than on graphical gimmicks.
- **Decisions:** As we want a scalable, cross-platform application, we decided to create a web application. This means most of the UI will be written in HTML5/CSS/Javascript. As the architecture is very important, our main focus the first few weeks will be on it, so we can build an adequate basis for future additions.

5.2 Review with teaching staff (26/02/2015)

- **Present:** Everyone (Kiani Lannoye, Tim Deweert, Robin Praet, Jan Van de Vyver)
- **Goal:** Getting feedback on our progress.
- **Meeting notes:** Progress report has a generally adequate structure. Meeting notes are well made, but we should send them to the client so we can get feedback and we have a document we can base ourselves on when something goes wrong. We need to be sure to document every decision we take and what risks it brings together with what functionality we are not going to provide as a result of this decision. We have been given a lot of freedom by the client and this is why we need to take more action ourselves. We should provide multiple options which can be extreme cases, so we can get to know what the client actually wants.

- **Decisions:** After meetings with the client, meeting notes will be sent to the client for his approval. We will be more proactive and turn our freedom into an advantage.

5.3 Stand-up meeting (04/03/2015)

- **Present:** Everyone (Kiani Lannoye, Tim Deweert, Robin Praet, Jan Van de Vyver)
- **Goal:** Deciding on libraries we will use so we can get started on thinking about the architecture.
- **Meeting notes:** We will no longer use WebRTC as it is too limited in functionality. Instead, OpenCV has been introduced as a usable library, because we can actually access the video frame by frame. Also, because of the fact that OpenCV is only available for the C++ and Python languages, we chose to make a Windows application instead of a web application. This because it is easier to make a native C++ application (making the GUI work with the encoding and stegonagraphy parts), so we can focus more on the data hiding aspect of the project.
- **Decisions:**
  - OpenCV will be used instead of WebRTC.
  - A Windows application will be built instead of a web application.

5.4 Meeting with client about architecture (06/03/2015)

- **Present:** Kiani Lannoye, Robin Praet, Jan Van de Vyver
- **Goal:** Showing current architecture to client and communicating made decisions.
- **Meeting notes:**
  - Our architecture is largely what the client expected.
  - We can assume a very simple network (1 switch).
  - We have to start with a Minimum Viable Product. The video must have some text in it, noise is allowed in the beginning.
  - It is very important that the video does not lag! In that case we can decrease the quality.
  - Text can be put in multiple frames to obtain some redundancy.
  - Steganography is very important, this is the crucial part of the application.
  - We have been recommended to use x265 for the HEVC encoder.
- **Decisions:**
  - UDP is preferred over TCP (less overhead $\rightarrow$ speedup)
  - C++ will be used (no Python). Also WebRTC is too limited, hence OpenCV is used (which has more possibilities).
  - x265 will be used as an encoder.

5.5 Meeting with client about x265 encoder project

- **Present:** Kiani Lannoye, Tim Deweert, Jan Van de Vyver
- **Goal:** Fix technical difficulties with the x265 encoder in Visual Studio 2013.
- **Meeting notes:**
  - Build the library that was available from github as a library in Visual Studio (lib file or dll file)
  - Another possibility is to include the current code from the steganography and video chat branch into this project, as it will be easier to use the x265 code.

5.6 Review with teaching staff (26/03/2015)

- **Present:** Kiani Lannoye, Tim Deweert, Jan Van de Vyver, Robin Praet
- **Goal:** Getting feedback on our progress.
- **Meeting notes:** We are working individually, but everyone knows the status of the project and we should continue working this way. Next review is within about a month from this date, we are expected to deliver some major improvements to the project.

5.7 Several meetings with client (Fourth iteration)

- **Present:** Kiani Lannoye, Jan Van de Vyver and Tim Deweert
- **Goal:** Fixing some issues with YUV and the x265 encoder.
- **Meeting notes:** During this iteration, we were trying to integrate the x265 encoder into our project to be able to feed the encoder frames we read from the webcam. We had lots of troubles with configuration and parameters necessary for the encoder to work (think about several YUV formats, smaller resolutions to have a higher framerate, ...). This is why we hopped by at our client/HEVC expert a few times to get some input about what we might have been doing wrong.
- **Decisions:** We have made a decision to slightly modify the project scope, refer to the decisions part to read more in detail.

5.8 Review with teaching staff (30/04/2015)

- **Present:** Kiani Lannoye, Tim Deweert, Jan Van de Vyver, Robin Praet
- **Goal:** Getting feedback on our progress.
- **Meeting notes:** We are having a lot of problems integrating all parts. We will plan some meetings with our client/experts on the topics of encoding and decoding. The final presentation is next, so there is still a lot of work to be done.

5.9 Several meetings with client (Fifth iteration)

- **Present:**Tim Deweert, Kiani Lannoye and Jan Van de Vyver
- **Goal:** Asking some questions about the ffmpeg decoder.
- **Meeting notes:** During this iteration, we were trying to integrate the x265 decoder into our project to be able to feed the decoder x265nal_units we get from the encoder. We had lots of troubles with building the ffmpeg library and include the decoder into our project. This is why we hopped by at our client expert a few times to get some information about the ffmpeg codec.