**DRM with D3R**

Table of Contents

[**1** **Purpose** 2](#_Toc527979853)

[**2** **About Gstreamer** 2](#_Toc527979854)

[**3** **Block Diagram** 2](#_Toc527979855)

[**4** **ECDH** 3](#_Toc527979856)

[4.1 Module Definition 3](#_Toc527979857)

[4.2 Block Diagram 3](#_Toc527979858)

[4.3 API 3](#_Toc527979859)

[**5** **Key Generation** 5](#_Toc527979860)

[**6** **Demo** 6](#_Toc527979861)

[**7** **Git Address** 7](#_Toc527979862)

1. **Purpose**

To implement Neowine’s DORCA3\_RIM(D3R) on Raspbery Pi to create a DRM platform with Gstreamer.

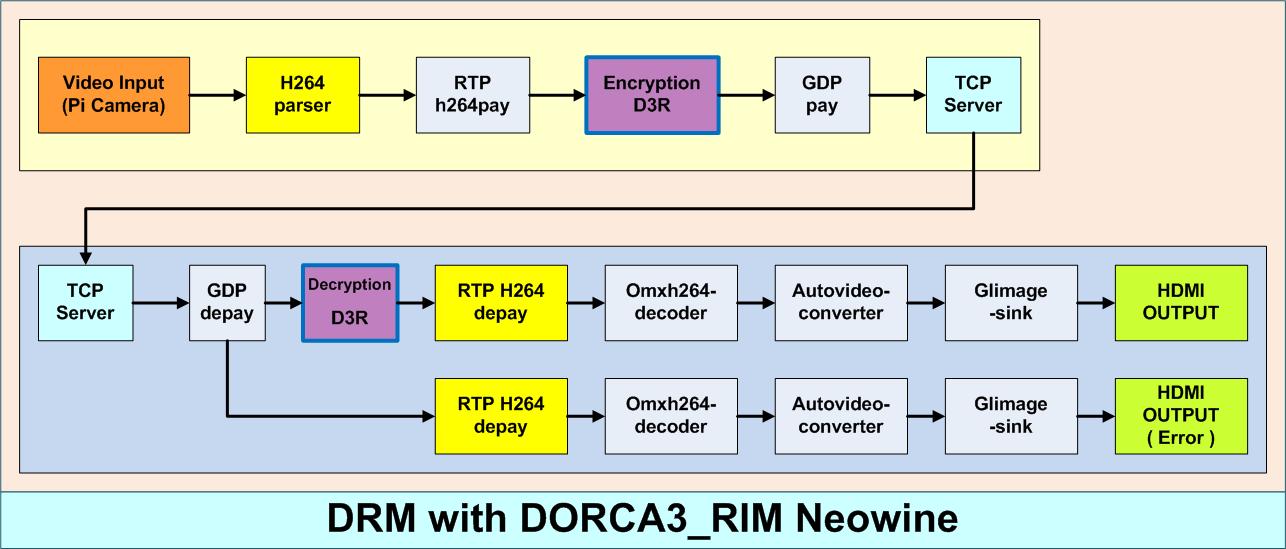
1. **About Gstreamer**

GStreamer is a [pipeline](https://en.wikipedia.org/wiki/Pipeline_(computing))-based [multimedia framework](https://en.wikipedia.org/wiki/Multimedia_framework) that links together a wide variety of media processing systems to complete complex workflows. For instance, GStreamer can be used to build a system that reads files in one format, processes them, and exports them in another. The formats and processes can be changed in a plug and play fashion.

GStreamer supports a wide variety of media-handling components, including simple [audio](https://en.wikipedia.org/wiki/Audio_frequency" \o "Audio frequency)playback, audio and video playback, [recording](https://en.wikipedia.org/wiki/Sound_recording_and_reproduction), [streaming](https://en.wikipedia.org/wiki/Streaming_media) and editing. The pipeline design serves as a base to create many types of [multimedia](https://en.wikipedia.org/wiki/Multimedia) applications such as [video editors](https://en.wikipedia.org/wiki/Video_editing), [transcoders](https://en.wikipedia.org/wiki/Transcoding), streaming media broadcasters and [media players](https://en.wikipedia.org/wiki/Media_player_(application_software)).

https://en.wikipedia.org/wiki/GStreamer

1. **Block Diagram**



The DRM platform has been implemented with Gstreamer pipeline. When sending the stream, using the method called probing, the source pad of GDPpay’s data is encoded, and when receiving the stream, the sink pad of GDPdepay is decoded using the same method.

The yellow box represents the server Raspberry Pi, and the blue box represents the receiver raspberry pi. The data transmission is performed via TCP/IP protocol. The data is first captured with Pi camera for the input, and is output by HDMI cable to a monitor. Encryption and Decryption are performed by D3R attached to both of the Raspberry Pis, and D3R could be set ON/OFF with a switch connected to the Pi’s GPIO.

1. **ECDH**

4.1 Module Definition

ECDH P256

4.2 Block Diagram



This module is used to create secure and common keys between two users. As it could be seen in the diagram above, USER A creates his/her own private key DA, and by perforing DA \* G, HA is created which is passed to USER B. USER B also creates his/her own pivate key DB which will create HB by doing DB \* G, and pass that to USER A. Finally, both USER A and USER B creates a common key called S by S = DAHB = DADBG for USER A and S = DBHA = DADBG for USER B.

4.3 API

**Function name: EcdhGenPubKey**

**Description: Receives public key by inputting secure key.**

**Parameter:**

uint8\_t\* sk[in] Private Key length: 32

struct \_point\* p1[out] Public KEy struct \_point {

uint8\_t x[32];

uint8\_t y[32];

}

**Function Name: EcdhGenPubKeyPuf**

**Description: Generates a random value, makes it into a secure key, and uses it to create a public key.**

**Parameter:**

struct \_point\* p1[out] Public Key struct \_point {

uint8\_t x[32];

uint8\_t y[32];

}

**Function Name: EcdhGenSessionKey**

**Description: Creates the final common key using the secure key from input.**

**Parameter:**

uint8\_t\* sk[in] Secret Key length: 32

struct \_point\* p1[out] Public Key struct \_point {

uint8\_t x[32];

uint8\_t y[32];

}

**Function Name: EcdhGenSessionKeyPuf**

**Description: Uses the secret key that was created before to create the final final common key. Ecdh\_gen\_pub\_key\_puff must be called before to create the secure key.**

**Parameter:**

struct \_point\* p1[out] Public Key struct \_point {

uint8\_t x[32];

uint8\_t y[32];

}

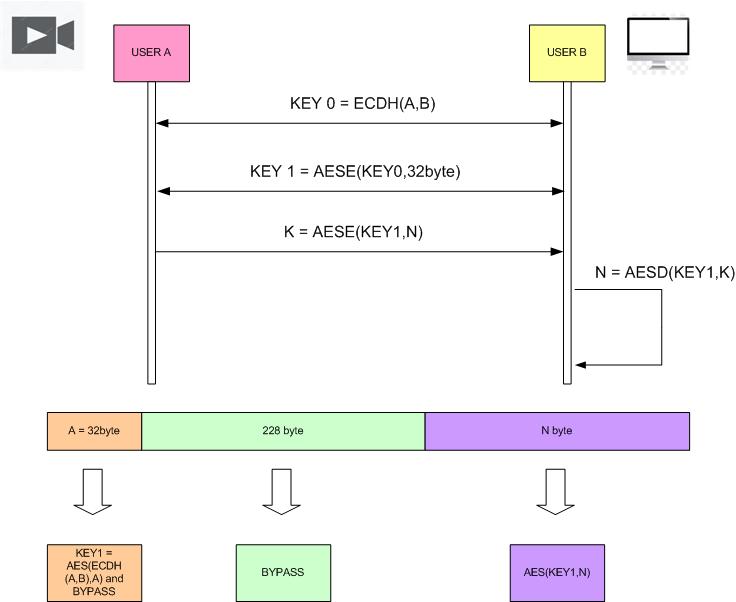
**Function Name: KeyFromEcdh**

**Description: Places the key obtained from ECDH into AES\_X0**

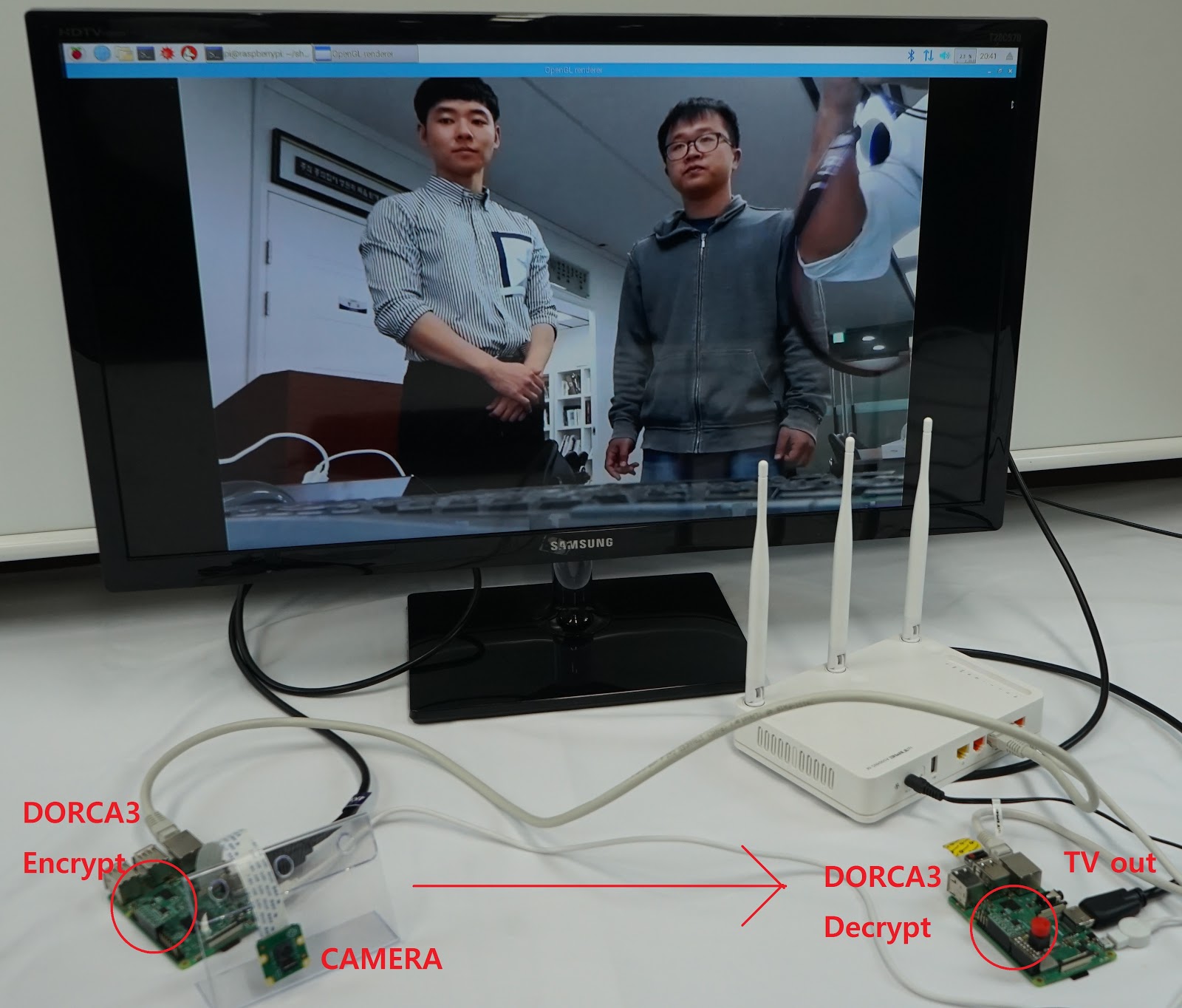
**Parameter:**

None

1. **Key Generation**

****

1. **Demo**

****

1. Prepare two Raspberry Pis and designate one as server and one as receiver.
2. Using the git clone command, download the necessary files from https://github.com/neowinepub/DRM\_D3R
3. For the server Pi, locate raspi\_streaming.c of function folder in gst-sender, and for the receiver Pi, locate raspi\_streaming\_player.c of function folder in gst-receiver, and correctly set the ip address to that of the sender pi.
4. If a switch is to be used to maneuver the ON/OFF status of the D3R of the receiver, correctly set the digitalRead’s input in cb\_have\_data to the GPIO that the switch is connected to.
5. Correctly attach the D3R in both server and receiver Pis, and attach the Pi camera on the server.
6. For the server, access the gst-sender and for the receiver, access the gst-receiver, and perform make in both machines and use the ./test command. (It is important to start the server Pi first.
7. For the sake of convenience, .sh file has been made for both receiver and sender scripts. After performing make once for each of the Pis, the DRM\_D3R could be executed by ./sender.sh or ./receiver.sh commands correspondingly in the DRM\_D3R folder.
8. **Git Address**

http://github.com/neowinepub/DRC\_D3R