**Countermeasure Against Deepfake**

Kyle Corcoran

Abdullahi Mohamed

Jacob Ressler

Team 7

Department of Electrical Engineering and Computer Science, Cleveland State University

Submitted to:

Dr. Ye Zhu

Associate Professor, Department of Electrical Engineering and Computer Science

# Table of Contents

[**Table of Contents**](#_u7jrv7483vpf) **2**

[**Executive Summary**](#_ypa7r83xws6p) **3**

[**1 Statement of Problem and Background**](#_ewnvyus2p13o) **4**

[**2 Design Objectives**](#_5wy4sy56c7xa) **4**

[**3 Technical Approach**](#_sh0g7vhnli4l) **5**

[3.1 Identifying the Unmet Needs](#_s5yx40vnqdms) 5

[3.2 Determining the Design Constraints](#_iftomq3dm4kr) 5

[3.3 Defining Technical Specifications](#_hcn0zclizuwg) 5

[3.4 Enumerating Design Concepts](#_wrhbgz4erd89) 6

[3.5 Selecting Design Concepts](#_w85krtaavy0i) 6

[3.6 Standards Compliance](#_c45g5dkw4di6) 7

[**4 Detailed Design**](#_gjdgxs) **7**

[4.1 FaceSwap Design](#_kgz0beijik3p) 7

[4.2 Steganography Design](#_30j0zll) 9

[**5 Verification**](#_1fob9te) **11**

[5.1 Experiment 1](#_mx3doyeazm2m) 11

[5.2 Experiment 2](#_b0vq28s8yoqr) 12

[5.3 An Issue with Facial Detection](#_lb8de2t5ntj9) 14

[**6 Project Management**](#_gole5c7aib8n) **15**

[6.1 Team Qualification](#_wvp3w8kh36rb) 15

[6.2 Task Assignment](#_yeifgw3ntarq) 15

[6.3 Timeline](#_sgwg3dozz5d3) 16

[6.4 Deliverables](#_t3i9rl5n8kex) 17

[6.5 Budget](#_j4kb3dmta13m) 17

[**7 Professional Awareness**](#_lxaa7n2j09ca) **18**

[**References**](#_t2mf00qsfwof) **19**

[**Appendix A: Résumés of Team Members**](#_p8z6e64q8enw) **20**

# Executive Summary

Determining real media from fake media is becoming increasingly difficult with the manipulation methods implemented in today's technology. These implementations make it possible for consumers with no special skills to alter digital media and create realistic synthetics using deepfake software available on the internet. Malicious actors are using machine learning to manipulate images to appear authentic to the human eye. While there exist some methods for detecting these “deepfakes,” none are perfect. As deepfake technology continues to advance at a rapid pace, there is a constant need to develop new methods to counteract its use. Physical copies of authentic items like signed baseball memorabilia use a certification to identify the authenticity. This paper proposes using steganography to embed a signed watermark inside a digital image. By using RSA to generate, sign, and verify the watermark, individuals will be able to authenticate personal images or try to verify signed images for authenticity.

# 1 Statement of Problem and Background

Deepfakes are an emergent and rapidly-developing technology that uses machine learning to generate synthetic media that replaces a person in a video or image with someone’s likeness. This technology was once very expensive but has become much more accessible and powerful over the years it has existed. Malicious actors have taken advantage of this to generate believable fake media for various unethical purposes such as deception, blackmail, and mischaracterization. In 2019 thieves used deepfake technology to automate the voice of a CEO and approve a wire transfer of more than $200,000. Incriminating videos have also been generated to extort and blackmail celebrities and politicians. As deepfake technology continues to improve, it is only logical to expect the misuse of it to increase. While there are already some countermeasures in place, they are quickly becoming inadequate. Digitally signing images will allow users to authenticate specific images. Using a verification method, users will be able to check the authenticity of signed images. This will improve detection of media manipulation and help deter malicious actors from using deepfakes to face-swap images.

# 2 Design Objectives

The goal of this project is to implement deepfake techniques with existing tools, design methods to detect fake content generated by deepfake techniques using FaceSwap, develop countermeasures to detect fake content generated by deepfake techniques.

* Use cryptography.py to create asymmetric keys to encrypt and decrypt our watermark.
* Use steganography to embed our water mark inside the input image.
* Download/Install deepfakes/faceswap open source software to begin experiments
* Successfully complete a deepfake to determine results on countermeasures and Implement counter measures

Cryptography.py requires Python 3.6+ and will be used to implement our RSA objects. This library will be used to create asymmetric private and public keys for signing and verifying images using steganography.

Steganography would be implemented using Python 3.9 and open source libraries. To extract pixel data from images in Python this project uses Pillow.py or PIL for image processing [6]. Numpy.py is used to store the image pixel bitmap inside a NumPy object for multi-dimensional manipulation [7].

Open Source software called FaceSwap will allow the demonstration of deepfakes. This software requires at least a powerful Central Processing Unit (CPU) or a Graphical Processing Unit (GPU). CUDA and plaidML are supported to enhance GPU performance which will help in shortening the time to train the data sets [2].

Deepfakes generated with FaceSwap are used to test our implemented steganography software. The software uses RSA objects to sign and verify for authenticity. We generate metrics on how successful design methods are.

# 3 Technical Approach

## 3.1 Identifying the Unmet Needs

Cryptography is used for securing the message, steganography is used to hide the message. This paper proposes combining the securities by securing and hiding the watermark of an image to simulate certificates. Currently there is not media forensic software that implements this process. This approach will allow users to secure and authenticate personal images and help deter malicious attackers from using authenticated media.

## 3.2 Determining the Design Constraints

Steganography is not perfect and there are some constraints to consider. Currently the implementation of this software will only work with still images. The design of this software is targeted for detecting deepfakes/faceswaps. So the algorithm is more accurate when the individual facial region is manipulated in the image. To generate our datasets of deepfake/faceswap required two NVIDIA Quadro P2200 GPUs.

## 3.3 Defining Technical Specifications

The technical specifications for the open source software FaceSwap are listed in the *README* file under “Required Hardware” [2]. The implementation of steganography is using Python 3.9 and the libraries mentioned in the design objectives.

## 3.4 Enumerating Design Concepts

|  |  |
| --- | --- |
| Cryptography | Uses RSA object to generate asymmetric private and public keys. With the private key a watermark can be signed. Signatures refer to an asymmetric property of the private key. The recipient of the signed watermark will be able to verify the signature is authentic with the asymmetric public key [9]. |
| Steganography | Steganography is the process of hiding information inside of other information [8]. This software implements least significant bit (LSB) steganography by reading in an image into a pixel bitmap. Then alternating the least significant bits of individual pixels to mask the signed watermark inside of a photo. The output would be the same image but with slightly altered pixels. The change to the LSB is too small for the human eye to detect. The reverse algorithm can be used to receive the masked message and flip the LSB back to the original values. This output will return the original image with a secret message. |
| Facial Detection | Facial Detection identifies the bounds of the face in the image and returns the coordinates. Using the coordinates Steganography can target those bounds and store the watermark in the facial region. Improving the accuracy of detecting deepfake/faceswap. |

## 3.5 Selecting Design Concepts

These concepts were selected by evaluating physical certificates of authenticity. Certificates are kept with the original item they are authenticating, so to implement this we chose steganography to embed a watermark into the image. Malicious actors would be able to use simple histograms [8] to find the embedded message and extract it. To combat this threat cryptography is used to sign the watermark with a private asymmetric key. If the watermark is manipulated then verification will not be possible. This idea is equivalent to a forged signature using asymmetric properties to identify the sender with the public key. Finally deepfake/faceswap was selected to test our dataset by authenticating an image with signing and verification.

## 3.6 Standards Compliance

There are already two bills that have been passed that address deepfake. In Virginia, criminalizing nonconsensual deepfake pornography, and one in Texas, criminalizing deepfakes that interfere with elections. Since deepfake has been an ongoing problem especially for celebrities, the government has been passing laws to address the issue. We just have to comply and not use any images of celebrities, especially not something that either affects or attacks an individual. As far as industrial standards I know that internet service providers that enable computer access to individuals who create or distribute deepfakes that violate the prohibitions are not liable. For our project we will comply with all the standards and regulations, we will not use deepfake to harm anybody in any kind of way and we will not violate the prohibitions.

# 4 Detailed Design

## 4.1 FaceSwap Design

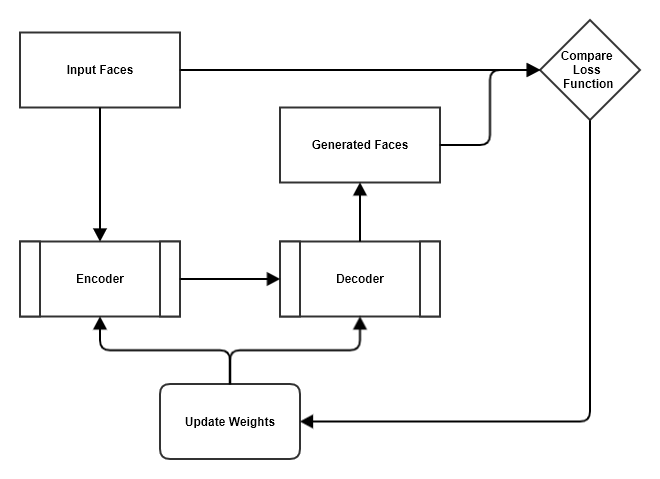
Faceswap is an open source program that allows the user to insert data from a source and target to train datasets. At a high level, training is teaching the Neural Network (NN) how to recreate a face. Most of the models are largely made up of two parts.

The NN needs to know how well it is doing encoding and decoding faces, which it accomplishes with two main tools.

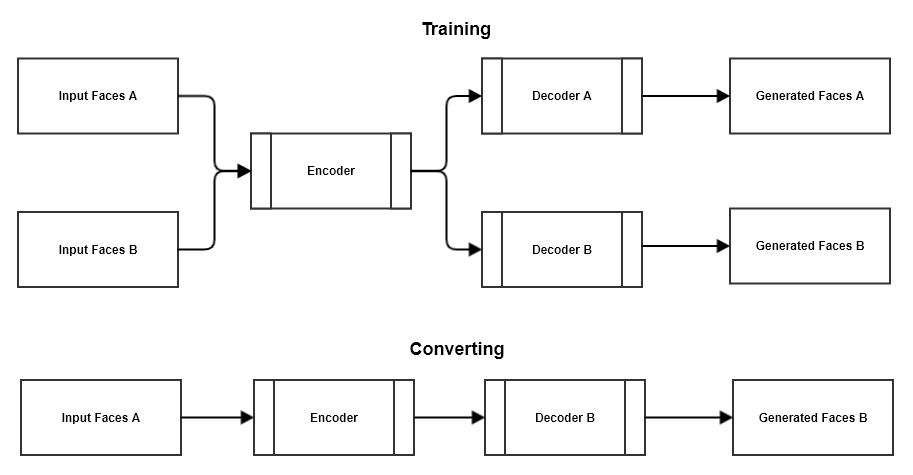
1. *Encoder* - This has the job of taking a load of faces as an input and "encoding" them into a representation in the form of a "vector".
2. *Decoder* - This has the job of taking the vectors created by the encoder and attempting to turn this representation back into faces, as closely matching the input images as possible.

The Neural Network needs to know how well it is doing encoding and decoding faces. So it uses two main tools to accomplish this. (see Figure 1)

1. *Loss* -  For every batch of faces fed into the model, the NN will compare current encoding and decoding algorithm to the actual face that was fed in. Based on how well it does, a loss value will be calculated and update the weights.
2. *Weights-* These are used to adjust the encoder and decoder algorithms. If the loss function is evaluated as getting more accurate, then the weights will continue to move in that direction. Otherwise, the weights will move in the other direction.

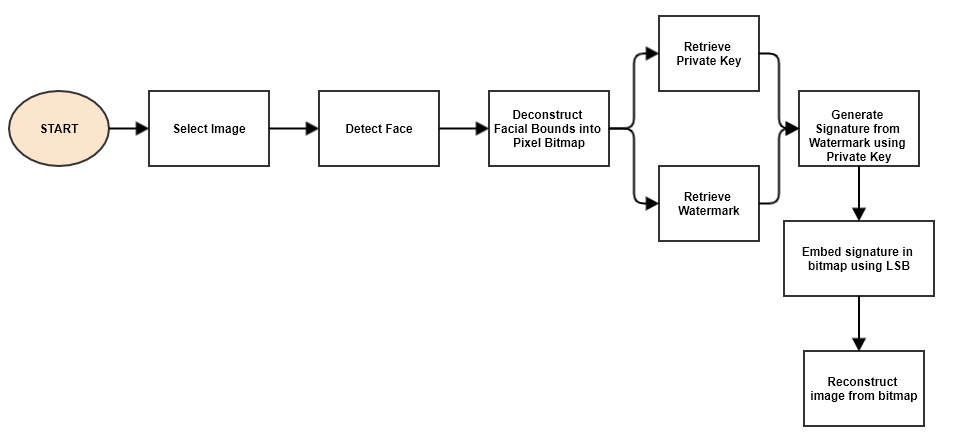
  
*Figure 1. Training with encoder and decoder*

The model then repeats this action a high number of times, constantly updating its weights based on its loss values, theoretically improving every time, until it reaches a point where the model has “learned” enough to effectively recreate a face, or the loss values stop falling. Once the database has been trained a shared encoder will be used to learn both sets of the faces required for the swap. Finally a switched decoder will be used to swap the face resulting in the swapped face being output in the Model (see Figure 2).

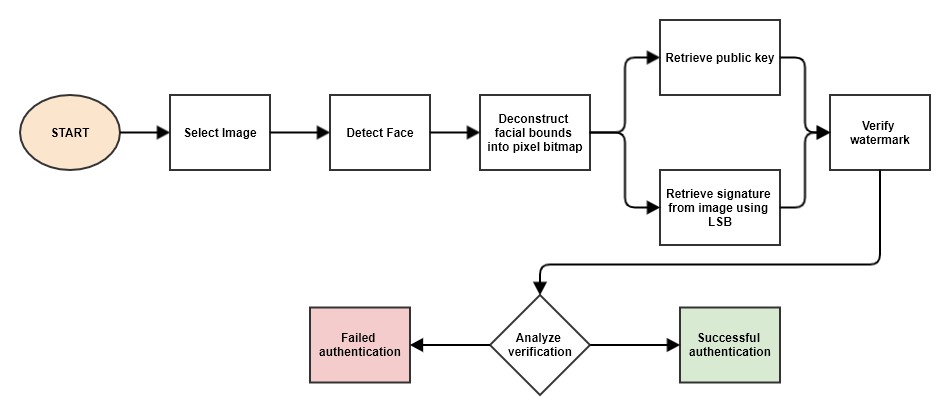
  
*Figure 2. Converting with encoder and decoder*

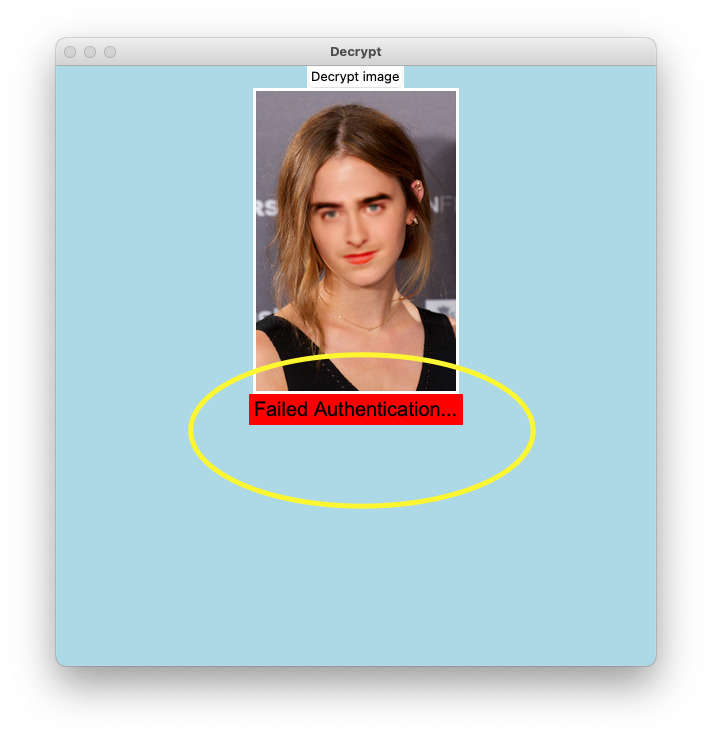
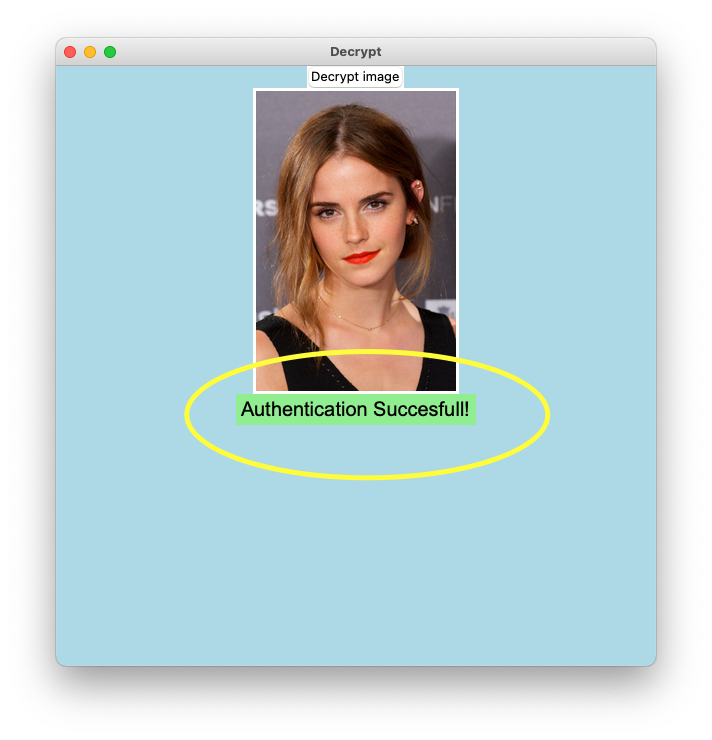
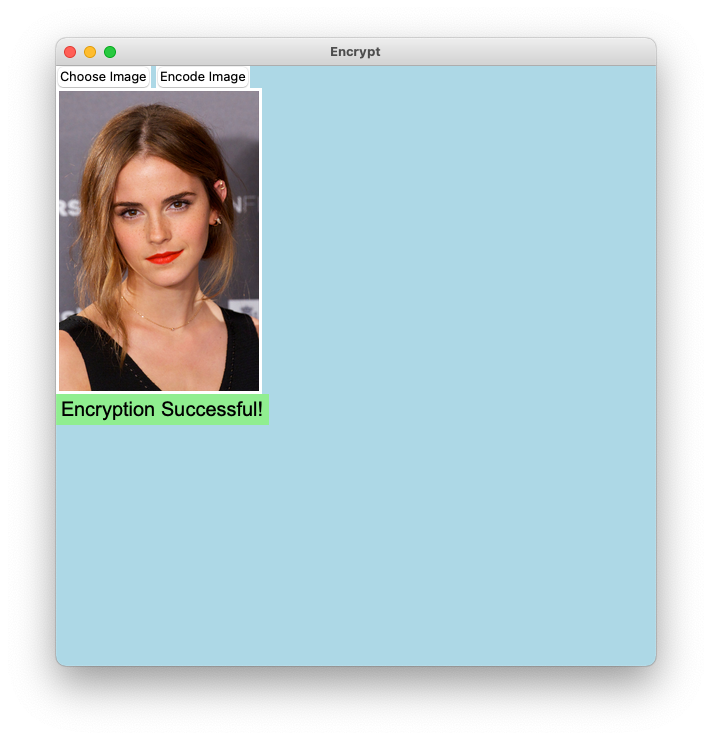
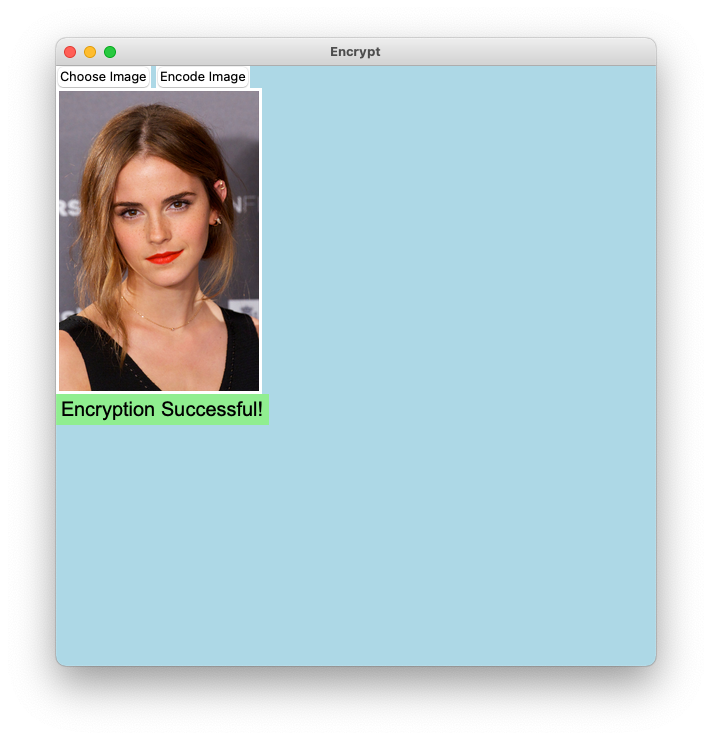
## 4.2 Steganography Design

To identify if an image has been deepfaked we will be using steganography to hide a signed watermark. This will be accomplished by writing a Python program. The user will run the program and select an image they wish to have signed. The image chosen will get stripped down into a pixel bitmap represented by binary bits. These bits will be stored into a string for manipulation. The watermark would then be deconstructed into a similar string and signed by a private generated key. The signed watermark would then be embedded into the pixel bitmap by altering the least significant bit (LSB) of each pixel [8]. Facial detection is used to identify the face and return the bounds of its pixels. A filter is applied to the bounds and the LSB of every pixel in the bounds is used to embed the watermark. The new bitmap is then used to reconstruct the image. This effectively “hides” the watermark from the naked eye, but it would be able to be recovered by using a verification software (see Figure 3).

*Figure 3. Flowchart of the signing process*

The verification software will be another Python program. The user will enter an input image they wish to authenticate. The image chosen will get stripped down into a pixel bitmap represented by binary bits. These bits will be stored into a string for manipulation. Using an asymmetric public key, the string will be verified to see if the private key signed the watermark. Verified images will be deemed as successful authentication, whereas unverified images will be deemed as failed authentication (see Figure 4).

*Figure 4. Flowchart of the verification process*



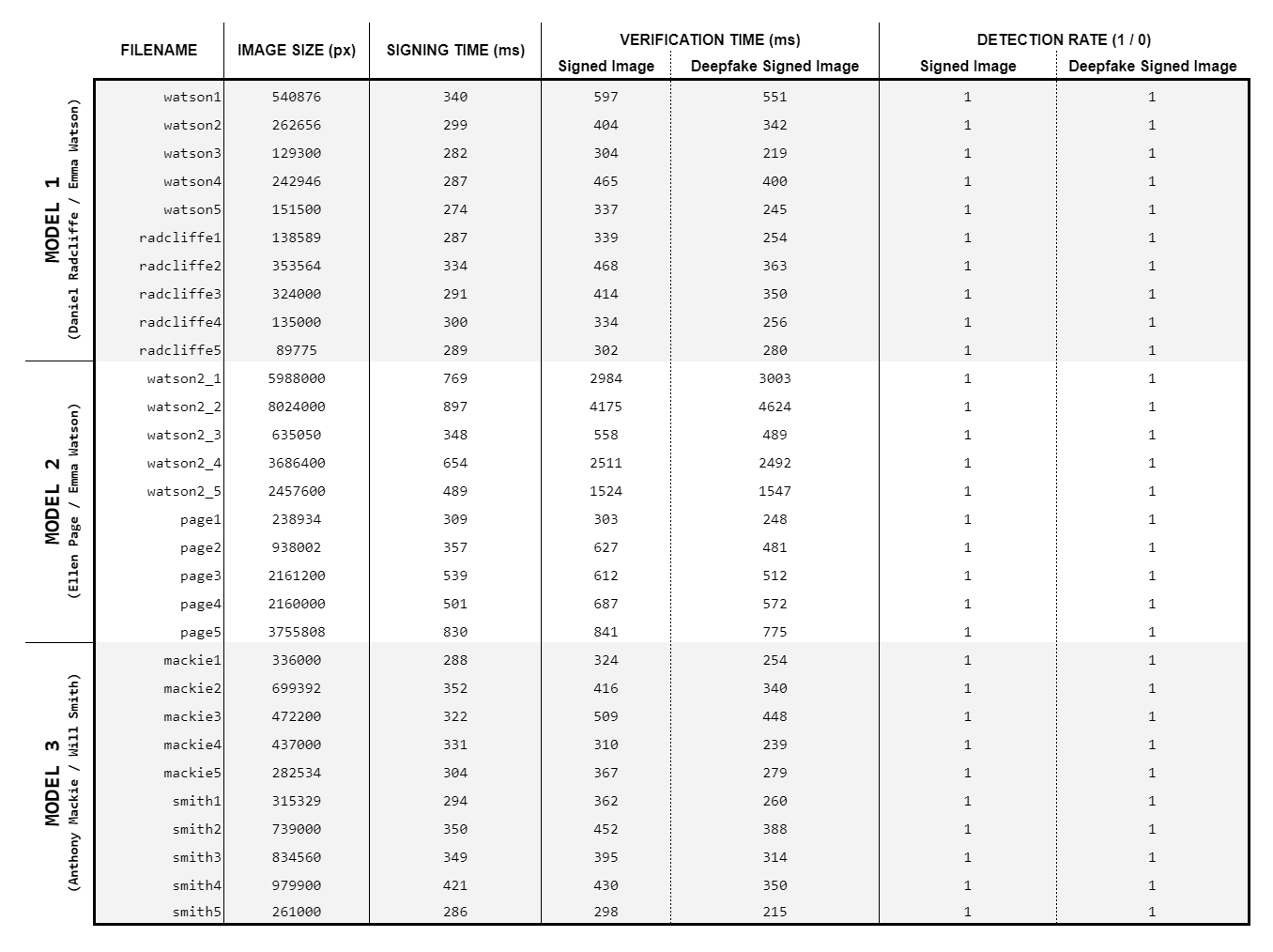
*Figure 5. Screenshots of failed and successful authentications*

# 5 Verification

Our software was tested on two general metrics: security and usability. Security is measured through detection rate, which is whether our image verification software is able to accurately detect both real and fake images. Usability is measured through execution times for the signing and verification processes. We performed two different experiments to test these metrics, with results shown in *Table 1* and *Table 2* respectively.

## 5.1 Experiment 1

In our first experiment, we selected 30 different images – 10 for each of our 3 FaceSwap models, split evenly between the two faces trained on by each model. *Model 1* was trained on Emma Watson and Daniel Radcliffe for 400,000 training iterations using FaceSwap’s *Original* trainer; *Model 2* was trained on Emma Watson and Ellen Page for 150,000 iterations using FaceSwap’s *Dfl-H128* trainer; and *Model 3* was trained on Anthony Mackie and Will Smith for 60,000 iterations, once again using FaceSwap’s *Dfl-H128* trainer. These variations were made to cover a general breadth of potential scenarios and subjects that could be encountered in the wild (differences in skin tone and complexity, gender, facial hair, overall quality of a deepfake, etc.).

*  
Table 1. Experiment 1: Security and usability metrics for variable image   
sizes using a static message*

Five pieces of information were recorded for each iteration of the experiment.

1. *Filename* – name given to base (unsigned, real) image
2. *Image Size* – total number of pixels in base image
3. *Signing Time* – time in milliseconds to create signed image from base image
4. *Verification Time* – time in milliseconds to verify 1) signed image and 2) deepfake signed image
5. *Detection Rate* – whether verification correctly (indicated by 1) or incorrectly (indicated by 0) detected 1) signed image and 2) deepfake signed image

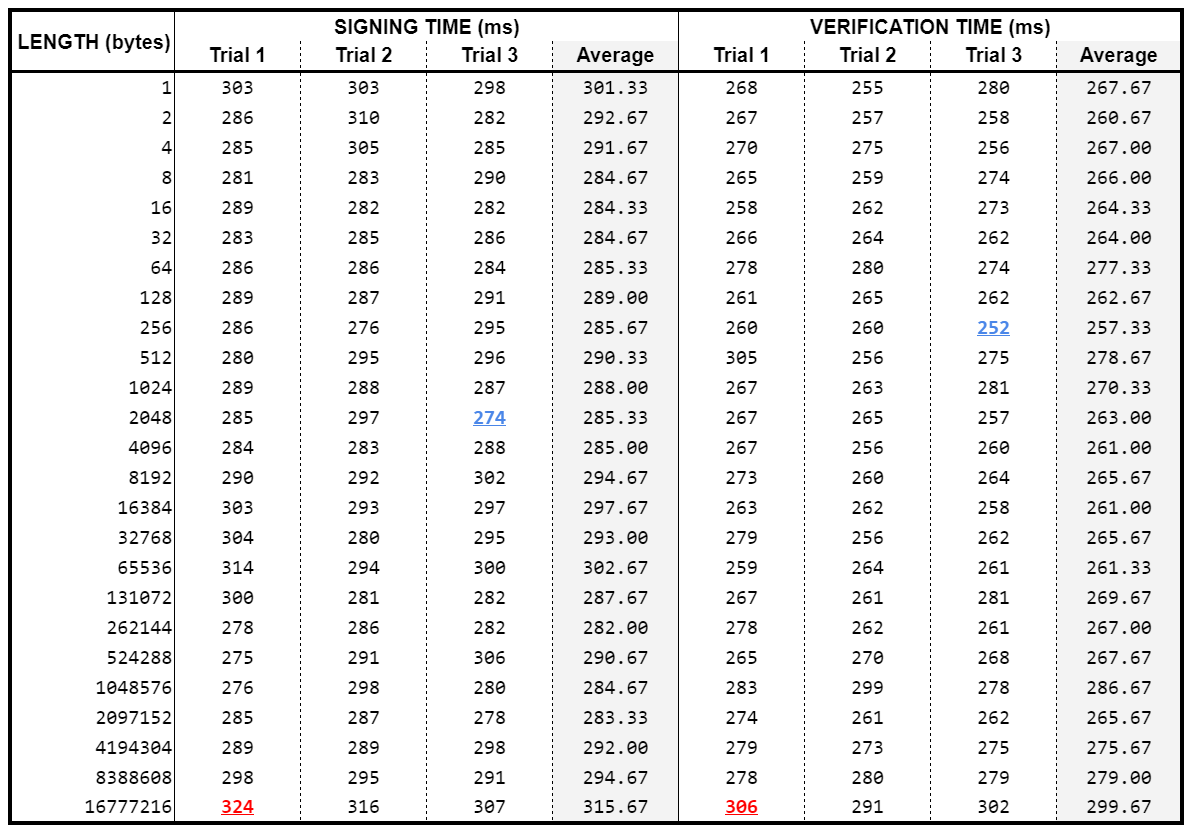
The experiment procedure is as follows. We start with a base image, indicated by *Filename*. This image is then signed using our signing software, with the *Signing Time* recorded. We take this new signed image and create a deepfake of it using FaceSwap and the respective model. This is our deepfake signed image. From there, we run both the signed image and the deepfake signed image through our verification software. The *Verification Time* and *Detection Rate* are recorded for each. The results of this experiment can be seen in *Table 1.*

From these results, we can see that image size directly affects both signing and verification times. Our largest image (*watson2\_2* which consists of just over 8 million pixels) took 897 ms to sign, 4175 ms to verify as a signed image, and 4624 ms to verify as a deepfake signed image. In contrast, our smallest image (*radcliffe5* which consists of just under 90,000 pixels) took 289 ms to sign, 302 ms to verify as a signed image, and 280 ms to verify as a deepfake signed image. The main reason for this is that both our signing and verification software utilize pixel traversal in their facial detection as well as their steganographic encoding (signing software) and decoding (verification software). As such, the more pixels an image has, the longer it will likely take to perform these specific pixel-based operations on said image. Some of this could be mitigated through further optimizations to our LSB algorithm, specifically regarding decoding.

Another conclusion we can reach from our experiment results is that our software is consistently able to correctly detect both signed images and their corresponding deepfakes. The verification software had a 100% detection rate for both signed images and deepfake signed images during our experiment. That is to say, all signed images were confirmed as being authentic, and all deepfake signed images were confirmed as being inauthentic.

## 5.2 Experiment 2

In our second experiment, we tested the effect of different message lengths on our signing and verification times. We began with a message length of 1 byte (a single character) and doubled the length with each iteration for 25 iterations. All iterations were performed using the same image (*page1* from Experiment 1). Verification times recorded are only for signed images, as no deepfakes were generated for this experiment.



*Table 2. Usability metrics for a constant image using variable message lengths*

Three pieces of information were recorded for each iteration of the experiment.

1. *Length* – length of the message in bytes (equivalent to number of characters in message)
2. *Signing Time* – time in milliseconds to create signed image from base image
3. *Verification Time* – time in milliseconds to verify signed image

The experiment procedure is as follows. We start with the base image *page1*. The base image is then signed using a signature generated from a message of the specified *Length*. The *Signing Time* of that process is recorded. The signed image is then verified using the same message as was used for signing, with the *Verification Time* recorded. This whole process is done three times for each message length, with the averages also recorded. All results for this experiment can be found in *Table 2.*

From the results, we can see a gradual general upward trend in both signing and verification times, though we believe some further testing would need to be done using larger message lengths so as to fully grasp the effect of message length on signing and verification times. If we account for randomness, all completion times up through the 8 MB mark are relatively similar. The average signing times from 1 B up to 8 MB have a range of 20.67 ms (from 282 ms to 302.67 ms), and the average verification times over the same interval have a range of 21.67 ms (from 257.33 ms to 279 ms). In both cases, however, there is a notable jump in completion time at 16 MB. Signing took an average of 315.67 ms to complete, and verification took an average of 299.67 ms, both of which are considerably greater than any prior times recorded over the course of our experiment.

We can logically assume the trend of increasingly more noticeable completion time differences would likely continue upon further doublings. However, as stated, more testing would need to be done in order to solidify this assumption.

## 5.3 An Issue with Facial Detection

After our experimentation, we encountered an interesting issue with our implementation of facial detection that we feel is novel and worth discussing in some detail. Very rarely, our verification software would fail to authenticate a signed image. We found that the source of this problem was due to the application of LSB manipulation in the facial region during signing. The minor changes made to the facial pixels would sometimes be enough to slightly shift the facial bounds returned by our facial detection. This would lead to a misalignment between where the signing software encodes the signature and where the verification software checks for the encoded signature, resulting in failed authentication.

There are a few potential solutions to this problem that we have come up with. One solution is to add a start sequence to our signature before signing our image. By doing such, we would be able to use slightly broader facial bounds within our verification software to search for said start sequence. The increase in facial bounds is simply to ensure that if there is a start sequence in the image, we will be able to find it. However, this approach could still result in the failed authentications of signed images if the width of its facial bounds is not equal to that of the base image’s facial bounds. If the width is less, we would miss some encoded pixels in our traversal. If the width is more, we would include unencoded pixels in our traversal. Both of these would result in incorrect signatures, and thus failed authentication.

Another solution would be to re-run our signed image through the facial detection algorithm to make sure it returns the same facial bounds as the base image. If it does not, we would create a new signature from our message and re-sign the base image. This would be done until the signed image and base image returned identical facial bounds. Such a solution could pretty heavily impact signing times (depending on how many attempts at signing would be needed), but it would in theory guarantee the authentication of undoctored signed images.

# 6 Project Management

## 6.1 Team Qualification

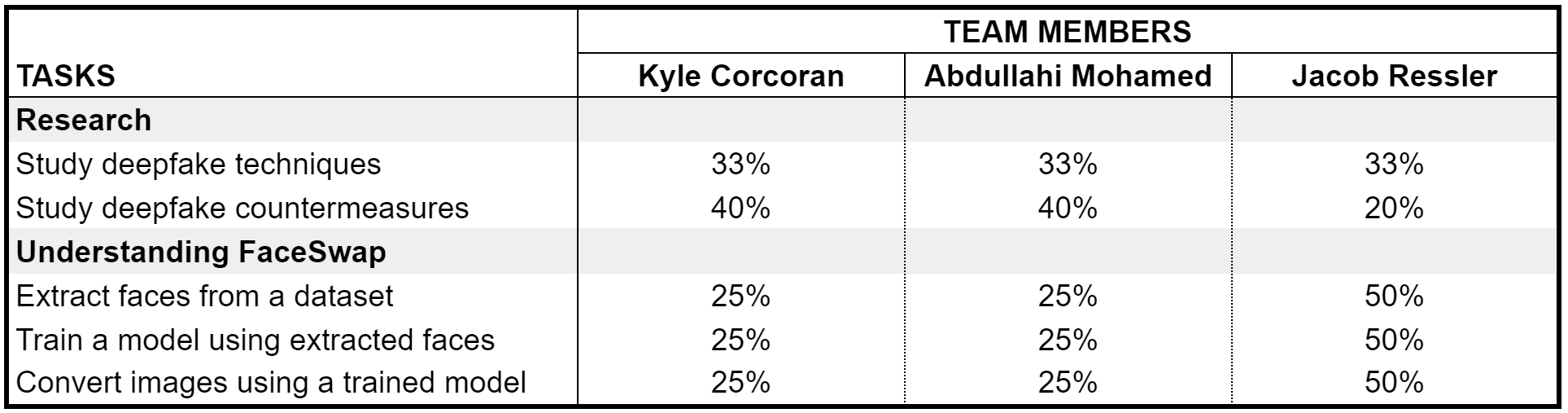
Kyle Corcoran is a senior-level Computer Science undergraduate student with good general knowledge of software and programming. He has experience with Python, a language used in the development stages of the project. He also has experience in technical writing, which has been greatly beneficial in compiling the process and findings of the project into a professional technical report.

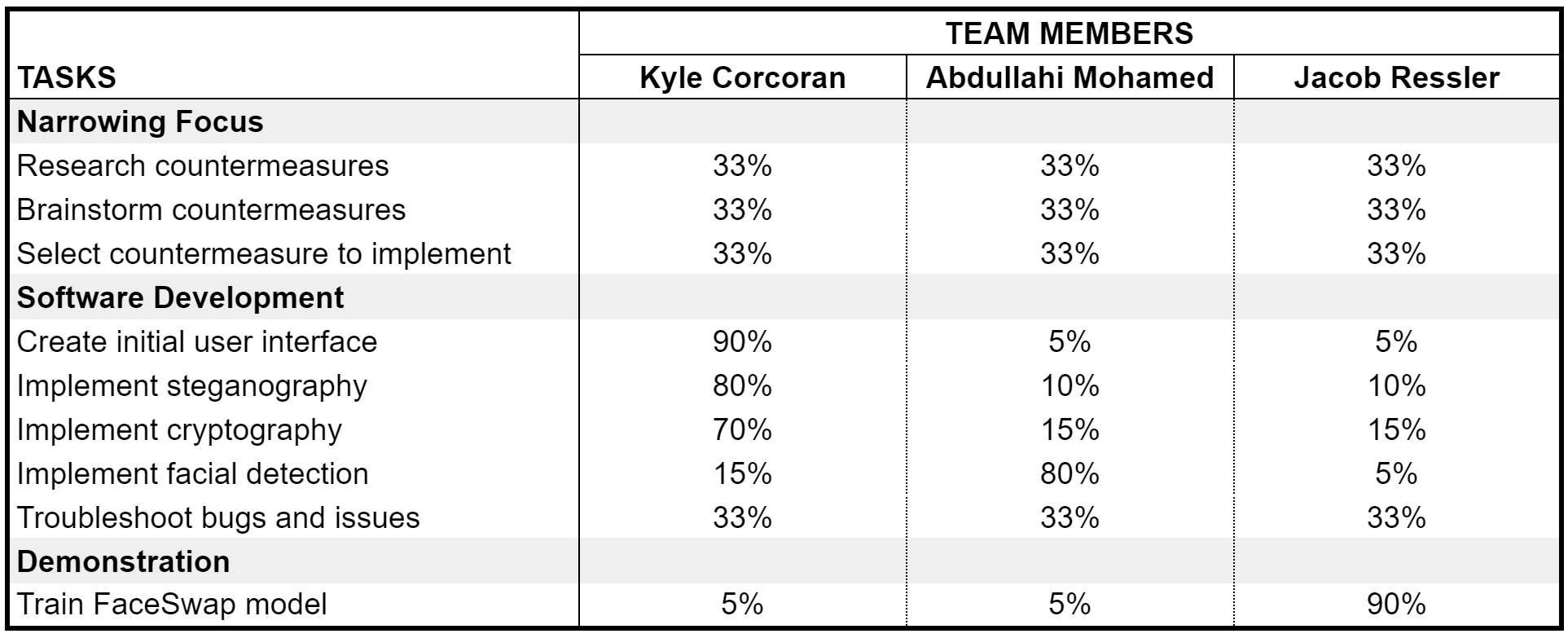
Abdullahi Mohamed is a senior-level Computer Science undergraduate student with good general knowledge of software and programming. He has experience in technical writing, which has been greatly beneficial in compiling the process and findings of the project into a professional technical report. He also has experience with Python, a language used in the development stages of the project.

Jacob Ressler is a senior-level Computer Science undergraduate student with good general knowledge of software and programming. He has some experience with Python, a language used during the development stages of the project. He possesses familiarity with the basics of artificial intelligence and machine learning, which has been beneficial in understanding FaceSwap.

## 6.2 Task Assignment

The following tables break down how the workload for this project was divided. As this project began very research-based, most work was shared by all members to various extents. As the project progressed and we got into more development-oriented work, it was divided up amongst us. Each member had their own areas to focus on, with any needed assistance provided by the other members so as to expedite the process. *Table 3* represents the tasks for the Fall Semester. *Table 4* represents the tasks for the Spring Semester.

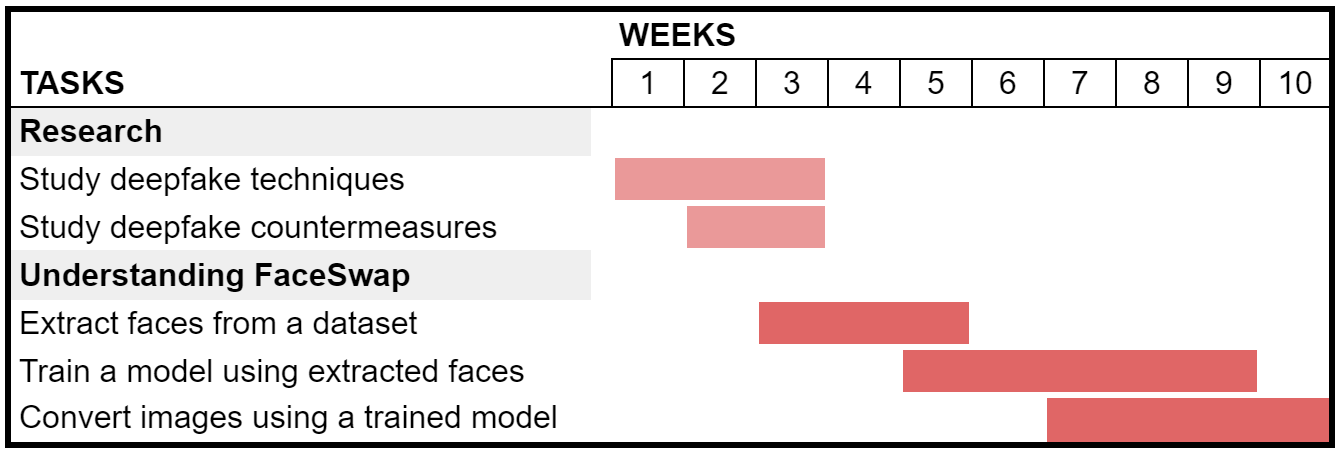
*Table 3. Task Assignments, Fall Semester*

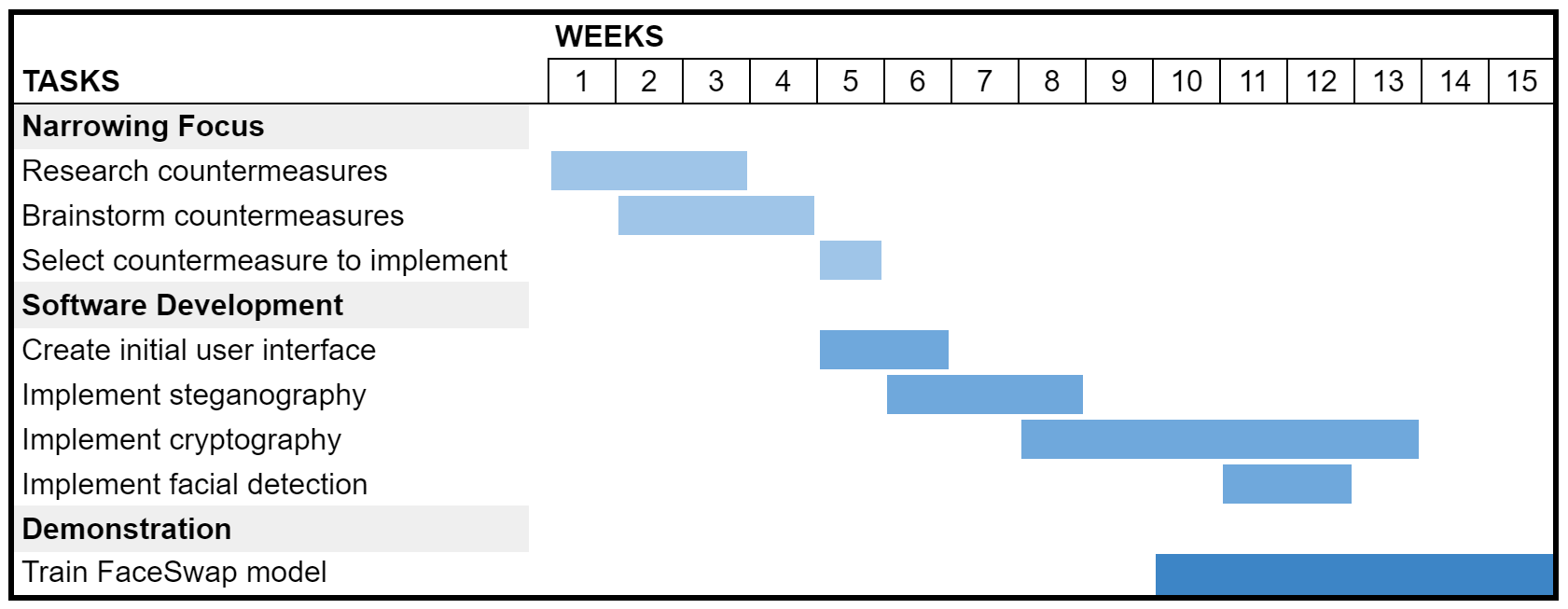
*Table 4. Task Assignments, Spring Semester*

Work for this report was divided up as follows. Kyle Corcoran handled the Executive Summary and Sections 1 through 3.3, along with Section 4, Section 7 and his resume for Appendix A. Abdullahi Mohamed handled Sections 3.4 through 3.6, and his resume for Appendix A, along with assisting Kyle in writing Section 4. Jacob Ressler handled Sections 5 and 6, along with his resume for Appendix A. The Table of Contents, Cover Page, and References were handled collaboratively by the team.

## 6.3 Timeline

The following Gantt charts map the timeline for this project. *Figure 6* represents the Fall Semester, with the primary tasks being to understand current deepfake techniques as well as their countermeasures and to become familiar with FaceSwap. *Figure 7* represents the Spring Semester, with the primary tasks being to narrow our focus to a single countermeasure of interest, implement said countermeasure as software, and create a demonstration showcasing the software at work.

*  
Figure 6. Project timeline, Fall Semester*

*  
Figure 7. Project timeline, Spring Semester*

## 6.4 Deliverables

This project has three deliverables. The first two deliverables are our signing and verification softwares. The signing software leaves a steganographic signature in an image. The verification software is used to verify that the signature is still intact on an image. The third deliverable is a demonstration of deepfake creation using FaceSwap and our signing and verification software. We will sign an authentic image of *Person 1*. We will then create a FaceSwap deepfake using the signed image of *Person 1* and a FaceSwap model trained on *Person 1* and *Person 2*. After that, we will use our verification software on both the signed image and the deepfake of the signed image to show that our program can detect the tampered signature in the deepfake as well as verify the authenticity of the signed image.

## 6.5 Budget

The project had a maximum budget of $500 with the only anticipated expense being the purchase of a GPU. We would have ideally liked to go with a cheap variant of either the NVIDIA RTX 2070 or the NVIDIA RTX 2060 Super due to their high CUDA core and Tensor core counts, along with their Turing architecture, all of which makes them very effective for machine learning processes like training a deepfake model. Unfortunately, due to the current shortage of GPUs in the market, graphics cards of that caliber were scarce, and those that were available were priced well beyond our $500 limit. As a result, we struggled to procure a GPU for this project, with the whole process taking much longer than originally anticipated. We ended up forgoing our original plan and instead choosing to purchase an NVIDIA Quadro P2200. Pricing details can be found in *Table 5*. While the Quadro P2200 was certainly less powerful than what we were hoping to get, it allowed us to utilize our workstation's current GPU, which was also a Quadro P2200. Even though the P2200 model does not support SLI (“scalable link interface,” allows multiple GPUs to be linked into a single output), we were still able to utilize both GPUs in FaceSwap through the program's support for distributed training and parallel processing.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Vendor** | **Quantity** | **Unit Price** | **Total (before shipping/tax)** |
| NVIDIA Quadro P2200 (5GB) | <https://www.cdwg.com> | 1 | $455.85 |  |
|  |  |  |  | $455.85 |

*Table 5. Purchase of an NVIDIA Quadro P2200*

# 7 Professional Awareness

As engineers developing this software we have a professional and ethical responsibility to the public. The software implemented is intended to detect, mitigate and prevent malicious actors from using deepfakes for heinous crimes.

The pre-existing techniques are being circumvented constantly by the malicious actors and require more research and learning to get ahead. This will be a lifelong process of constantly learning to detect better measures to prevent crimes with deepfakes. The software intends to restore popular trust in media authenticity and deter the spread of synthetic propaganda.

# 

# References

[1] L. Verdoliva, "Media Forensics and DeepFakes: An Overview," in IEEE Journal of Selected Topics in Signal Processing, vol. 14, no. 5, pp. 910-932, Aug. 2020, doi: 10.1109/JSTSP.2020.3002101.

[2] L. Guarnera, O. Giudice and S. Battiato, "Fighting Deepfake by Exposing the Convolutional Traces on Images," in *IEEE Access*, vol. 8, pp. 165085-165098, 2020, doi: 10.1109/ACCESS.2020.3023037.

[3] Torzdf (2019) deepfakes/faceswap (Version 2.0) [faceswap.py]. github.com/deepfakes/faceswap.

[4] T. Jung, S. Kim and K. Kim, "DeepVision: Deepfakes Detection Using Human Eye Blinking Pattern," in *IEEE Access*, vol. 8, pp. 83144-83154, 2020, doi: 10.1109/ACCESS.2020.2988660.

[5] Yisroel Mirsky and Wenke Lee. 2020. The Creation and Detection of Deepfakes: A Survey. ACM Comput. Surv. 1, 1, Article 1 (January 2020), 38 pages

[6] Lundh, F., & Clark, A. (n.d.). Pillow. Retrieved April 08, 2021, from <https://pillow.readthedocs.io/en/stable/>

[7] Oliphant, T. E. (n.d.). Numpy. Retrieved April 08, 2021, from <https://pypi.org/project/numpy/>

[8] D. Neeta, K. Snehal and D. Jacobs, "Implementation of LSB Steganography and Its Evaluation for Various Bits," 2006 1st International Conference on Digital Information Management, Bangalore, India, 2007, pp. 173-178, doi: 10.1109/ICDIM.2007.369349.

[9] "45 A Method for Obtaining Digital Signatures and Public-Key Cryptosystems (1978)," in Ideas That Created the Future: Classic Papers of Computer Science , MIT Press, 2020, pp.463-474.

# 

# Appendix A: Résumés of Team Members

The following pages present one-page résumés of the team members for this project. The order is as follows: Kyle Corcoran, Abdullahi Mohamed, Jacob Ressler.

**Kyle Corcoran**

99 W. St. Claire Ave. Apt. 208. Cleveland, OH 44113 | [k.e.corcoran@vikes.csuohio.edu](mailto:k.e.corcoran@vikes.csuohio.edu) | (706) 908-2179 | <https://github.com/Kyle-C-CSU>

**Summary:**

Former Diesel mechanic in the United States Marine Corps, currently a computer science student at Cleveland State University. Demonstrated skill in C and training in Java. Seeking a professional opportunity to excel in software development. Eager to contribute to team success through hard work, attention to detail and excellent organizational skills.

**Education:**

Cleveland State University (CSU), Cleveland, OH

Expected graduation: May 2021

**Honors or Awards:**

Navy Achievement Medal, United States Marine Corps

2016

**Skills:**

* Java
* C++
* C
* Troubleshooting
* Customer Service
* Procurement
* Leadership
* Communication

**Work Experience:**

BarFli Inc. Los Angeles, CA

2020-Present

Lead Developer

* Designed UML to organize the Object Oriented Programming (OOP)
* Implemented simple Graphical User Interface (GUI) for login page using Xcode’s Story Board
* Researched method for connecting application to a database

Stefanini, Southfield, MI

2017-Present

Service Desk Technician

* Analyzed, evaluated, and responded to hundreds of user incidents and inquiries using test scripts and other troubleshooting methods, resulting in increased customer satisfaction and meeting service license standards.
* Managed the CVC queue for pending request tickets to get procured in order for our clients to receive the hardware and software required.
* Followed up with clients to ensure optimal customer satisfaction following support engagement and problem resolution.

**Military Experience:**

United States Marine Corps, Camp Pendleton, CA

2012-2016

Sargent Diesel Mechanic | Line Mechanic | Crew Chief | Tool Room NCO

**Volunteer Experience:**

Volunteer, GiveCamp, Cleveland, OH

2019

Assisted various non-profit organizations with improving iOS applications using xCode.

**Abdullahi Mohamed**

UNDERGRADUATE STUDENT

[a.a.mohamed@vikes.csuohio.edu](mailto:a.a.mohamed@vikes.csuohio.edu) 3139 W. 98th St., Cleveland, 44102, United States 216-256-8285

**Education**

**Bachelor of Science, Cleveland State University, Cleveland**

August 2018 - May 2021

**Associate of Science, Cuyahoga Community College, Cleveland**

May 2016 - May 2018

**Skills**

Java

Python

BMC Client Management

Linux OS

C/C++

Footprints

Active Directory

SQL

**Technical Experience**

**IT Intern at Medical Mutual, Cleveland**

May 2018 - May 2020

* Analyzed the process of reimaging and refreshing machines
* Constructed Windows 10 machine from scratch and upgraded existing machine to Windows 10
* Developed problem solving skills through troubleshooting install/upgrade issues
* Resolved service tickets from every department in the organization utilizing the software Footprints and Provance BMC Client Management

**Infrastructure Engineer Intern at Medical Mutual, Cleveland**

May 2020 - Present

* Performed server related operational tasks
* Analyzed decommissioning process of Server 2012
* Assisted with service ticket resolutions and the assignment of server tickets for server team
* Experience working with Active Directory
* Gained overall understanding of the infrastructure team

**Accomplishments & Memberships**

**Chevron Corporate Scholar**

July 2020 - Present

**Honeywell IPP/NSBE Scholar**

April 2020 - Present

**Vice President of National Society of Black Engineering**

April 2020 - Present

**The National Society of Leadership and Success**

August 2019 - Present

**President of Somali Bantu Community**

September 2017 - Present

**Ernst & Young College Map Scholar**

September 2015 - Present

20968 WOODSTOCK AVENUE,

FAIRVIEW PARK, OH 44126

[JRESSLER96@GMAIL.COM](mailto:JRESSLER96@GMAIL.COM)

(440) 596-8773

[GITHUB.COM/JACOB-RESSLER](https://github.com/jacob-ressler)

JACOB RESSLERA long, thin rectangle to divide sections of the document

SKILLS

* Java
* JavaScript
* Teaching
* Working with children
* C#

EXPERIENCE

**Lutheran High School West - Rocky River, OH**

*Wrestling Coach*

NOVEMBER 2015 - PRESENT

* Volunteer Assistant Coach for Boys and Girls High School wrestling programs
* Assistant Coach for Jr. High School wrestling program
* Co-Head Coach for Youth wrestling program

**St. Thomas Lutheran School - Rocky River, OH**

*After-School Care Supervisory Aide*

AUGUST 2016 - JUNE 2019

* Homework room supervisor, grades 3-5
* Large motor activity assistant supervisor, grades K-5

EDUCATION

**Cleveland State University - Cleveland, OH**

*Pursuing a Bachelor of Science in Computer Science*

EXPECTED GRADUATION: MAY 2021

AWARDS & ACHIEVEMENTS

**Dave Kiraly Memorial Christian Sportsmanship Award**

2014, 2015

**Washkewicz College of Engineering Dean’s List**

AUGUST 2017 - JANUARY 2019, JANUARY 2021