CS 201 Group Project
Face Hasher

Henryk Sosnowski
Bryan Beus

November 25, 2019

Source Code Link: `https://github.com/sosnowskih/facehasher`
This project took approximately 6 hours to complete, up to this point in time.

1 Pitch

Our software application is a solution to a problem arising from ”Deep Fake” technology. Deep Fakes allow a malicious person to create a fake, but fully believable, visual history of another person. This can be used to target the victim with hoaxes that damage their reputation.

Google and governments are looking for ways to provably detect these hoaxes, but so far, no solutions have been found.

Our software application takes a different approach. Rather than try to detect the hoax after the fact, instead the potential victim prevents the hoax by securely recording their own bio signatures (as cryptographic hashes) into a secure blockchain, such as Bitcoin.

The blockchain technology now holds a decentralized record that proves the would-be victim's actions are not associated with the hoax.

2 Design

2.1 Overall Design

As deep-fake technology becomes prevalent in the lives of the general public, audiences will struggle to discern truth from falsehood. Some people may find themselves victims of hoaxes, or even targets of malicious attacks. Victims may suffer damage to their reputation, livelihoods, and general well being.

Our software is designed to empower users with the ability to prevent these attacks, through a combination of visual data records, cryptographic hashes, and blockchain technology. Our prototype software maintains its simplicity by being capable of running as a background process.

The dangers presented by deep-fake technology lend credibility to the notion that this type of software is becoming increasingly necessary. Without a hash stored in a blockchain that records your location and activities, users will find it impossible to prove where they were, what they were doing, and whether or not they were acting within the bounds of acceptable behavior.

Many types of computer devices, including desktops, laptops, phones, watches, and other devices; will include cryptographic hashing software by default as a means of allowing the user to protect themselves. Smart computer hardware vendors, likewise, will seek to capitalize on the financial profits potentially available using the blockchain-specific side of this application.

We seek to work on a prototypical solution to this problem because we are interested in the possible future applications of these concepts. Studying and implementing these concepts now can help us as we prepare for future career opportunities. Furthermore, these software applications allow us the ability to test our knowledge against an array of different useful libraries.

2.2 Prior Art

Two software movements inspired our software application. The first is the deep-fake technology popular on social networks, such

as Reddit. The second is blockchain-based record-keeping software, such as Factom.

The deep-fake movement became popular several years ago on Reddit. Popular applications of this technology included taking one's own face and the digital video of a popular (or memetic) movie scene, and applying machine-learning algorithms to recreate oneself as that movie actor within the story. [Lis19]

While these simple tricks were useful for common memes and harmless jokes, the usefulness of deep fakes for malicious purposes became clear as female celebrities found their likenesses being applied to risque film of a sordid nature (pornography). There is, as of yet, no method available for the female celebrities to prevent this type of film from being created, and as the technology improves, the public may not be able to discern between deep-fake falsehoods and reality.

The blockchain-based record-keeping software, Factom, made early use of blockchain technology and cryptographic hashes to create useful, provable records for business application. These were not specifically tied to Deep Fakes, but the concepts are effectively similar.

With Factom, a business owner requires that their employee create a hash of company data on a regular basis. For example, if the company expects to move fifteen supply truck's of apples from one city to another, the employee must record the data of the movement of these apples at all points in time. The data is then hashed and inserted into the Factom blockchain, creating a provable record of the time that this data existed. [Fac19]

This protects the business owner from a common danger that arises when an employee fails to perform their duties accurately. Without the blockchain record, and if we assume that the employee accidentally allowed the apples to be destroyed (or even stolen), the employee could potentially change the record within a centralized database to make the record appear as though the blame lay elsewhere. This could lead to confusion for the business owner, as he tries to sort out who made a mistake, and how the error should be resolved and prevented in the future.

With the blockchain record, however, the employee cannot go back in time, create a hash, and insert this hash into the blockchain

record. Blockchain technology is designed to secure the record against these types of attacks. Therefore, an employee's potentially harmful desire to falsify records is mitigated, while the business owner receives protection.

Likewise, we expect in time that users of all types will desire to have a record of data for themselves that ensure they are safe against malicious deep-fake attacks. The manner in which these attacks can proceed is similar to the manner of attacks in the supply-chain network. Malicious people create false histories, and users protect themselves against false histories by making digital, blockchainified records of their actions to prove the Deep Fakes false.

2.3 Technical Design

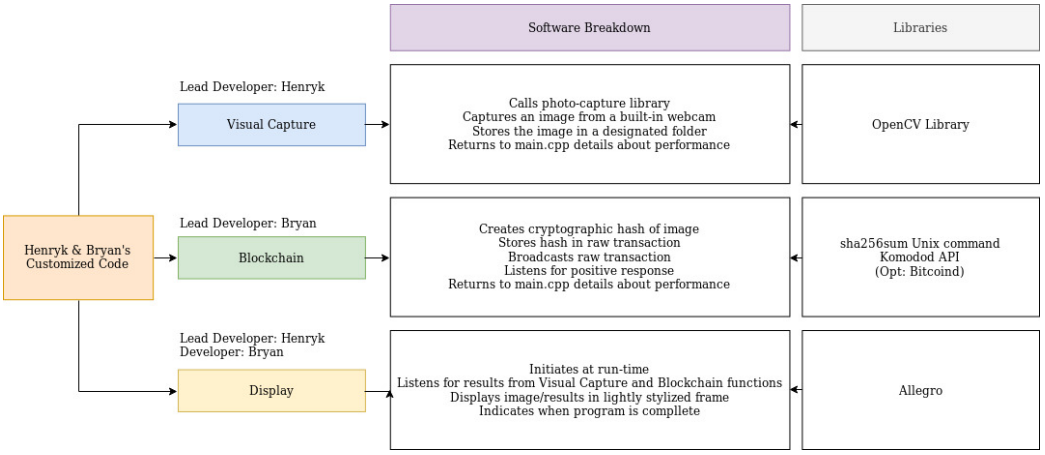

Figure 1: Software Architecture

Our software is divided into three separate portions. The first portion manages the camera API. The second hashes the image and manages the blockchainification of the hash. The third portion manages the visual display elements of our interface.

The Camera API code has a few separate tasks, and Henryk is the lead developer on this portion of software. The code first connects with the default webcam built into the computer hard-

ware on the host device. Once connected, the code acquires an image taken by the webcam. The code then saves this image to a designated folder, and returns the location and name of this file to the next portion of the software.

The blockchain portion of the software receives the location of the image, and begins the process of blockchainification. Bryan is the lead developer for this portion. The code first creates a hash of the image. The code then connects with the Komodod daemon's API, creates a raw transaction that includes the hash in the `OP_RETURN` field, and then broadcasts this transaction to the network. The code listens until the blockchain reports that this hash is confirmed on the network, and returns the performance results.

The display portion of the software is active throughout the program, and Henryk is again the lead developer here, with Bryan assisting once finished with the blockchain code. The display initiates on launch of the software, displays the photo taken from the webcam (once available), displays icons that indicate the progress of the software's goals, and shows a console readout that reports any alerts the developers choose to print.

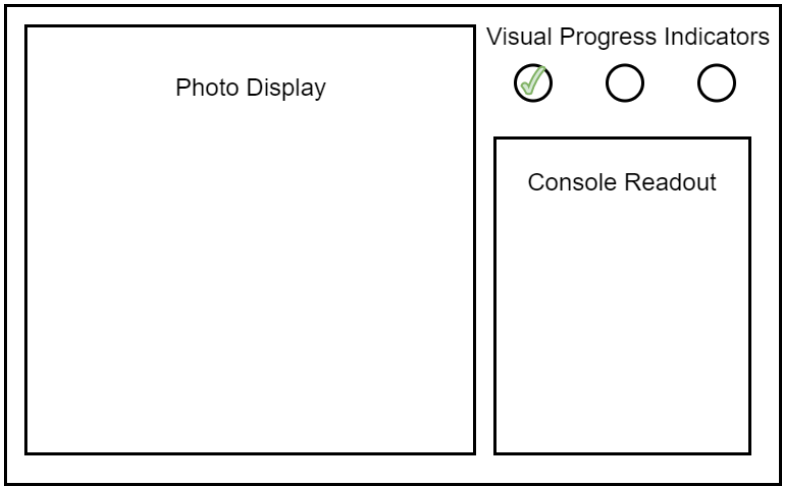

Figure 2: User Interface Design

Our GUI design intends to be simple and informative, without any need for interaction; the program runs only once, as an

intentional limitation to keep the scope of the project manageable. Our GUI display allows the user to see the image that is hashed on the left of the screen, with console readout on the right displaying program instructions and results. Above the console readout are blank/filled check boxes to serve as visual indicators of progress.

2.4 Required Libraries

- OpenCV Library
  - This library allows us to access the default webcam built into most computer hardware devices, and running all three of the major operating systems
- sha256sum Unix Command
  - This Unix command allows us to make a simple hash of an image, for blockchainification
- Komodo API (komodod)
  - The Komodo API allows us to create raw transactions that hold data in the `OP_RETURN`, and to monitor the confirmation of these transactions, and to verify the data's placement
- (Optional) Bitcoin API (bitcoind)
  - The Bitcoin API can, optionally, be used to verify the hash that is ultimately inserted from the Komodo blockchain into the Bitcoin blockchain
  - This ultimately allows the user to rely on the Bitcoin blockchain, and the data of their image and the blockchains, to prove where they were at a specific point in time
- Allegro
  - The Allegro UI library allows us to create a simple GUI display that shows the progress of our software

3 Project Update

3.1 Henryk Sosnowski

My goal for this stage of the project was to complete the image capturing portion of the overall program so Bryan will know what kind of output to expect when creating the hash. This functionality relies on the basic components of the OpenCV library to operate but is quite simple in its execution. First the function attempts to access the camera using the computer's default camera settings and returns an error if it fails for any reason. Next it opens a window displaying the live image being captured by the camera that remains until an appropriate key is entered by the user. When the image capture key is pressed, the current frame is saved as "image.jpg" to the program's root directory. The window will remain open until the user decides to end the process, so the image can be taken multiple times if desired, replacing the previous file.

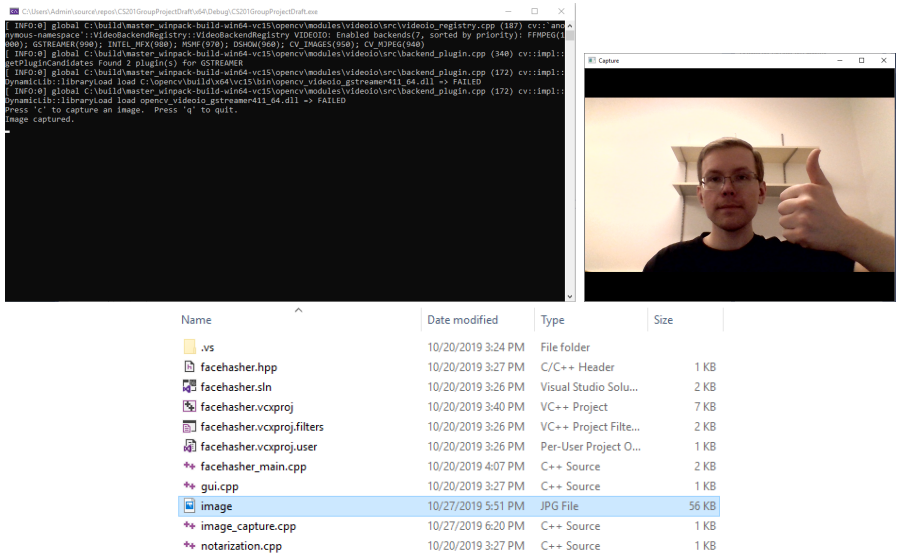

Figure 3: Henryk Sosnowski Artifacts

As seen in Figure 3, there are currently some messages printed

to the console by OpenCV functions that do not impact functionality. As we are planning on creating a GUI once the program's core functionality is complete, I chose to leave those messages as-is for now. The details of other portions of the image capture functionality will most likely be partially altered later in development for integration into a GUI, but it is functional for our needs in its present state.

3.2 Bryan Beus

To ensure that the program is manageable within the scope of the class project, my goal for this period of time was to implement all aspects of hashing and blockchainifying an image.

For the hashing aspect, I researched several different options. Some possibilities included using the `system()` C++ function, but this function supposedly has a bad reputation. The function is both time and memory expensive. I also downloaded a hashing library that I discovered through Google, and this looks to be a promising long-term solution, as the library claims to be cross-platform compatible. However, I was not able to implement the library on the first attempt, and so, for now, I abandoned this library for a simpler solution.

In the end, for hashing, the best solution I found was to use the `popen()` function for Unix-based hashing. This function is included in the standard library and allows me to both execute a command and capture the response as a string, which I can then manipulate as desired (including turning the string into a json object for data manipulation).

This is not an effective long-term solution, because `popen()` is only compatible with Unix machines. If we choose to continue with this function, we may need to implement a virtual machine on Henryk's machine, or we may need to create a docker image. Alternatively, we can try again later to implement a cross-platform compatible option.

Once I have captured the hash of the image, I then send this hash to the RICK Komodo-based blockchain for storage. The key challenge here was to connect my source files to the API of the

Komodo daemon. The API requires `curl` commands for connectivity, and I tried to implement a `curl` C++ library for this purpose. Again, I was not able to implement the library, and therefore I resorted to assembling the commands using string manipulation and the `popen()` function. The returned response from the API is a json object. For now, I have to capture this json object as a string, and then convert it back into a json object within my source files. This is cumbersome, and I intend to correct this before the project is finished.

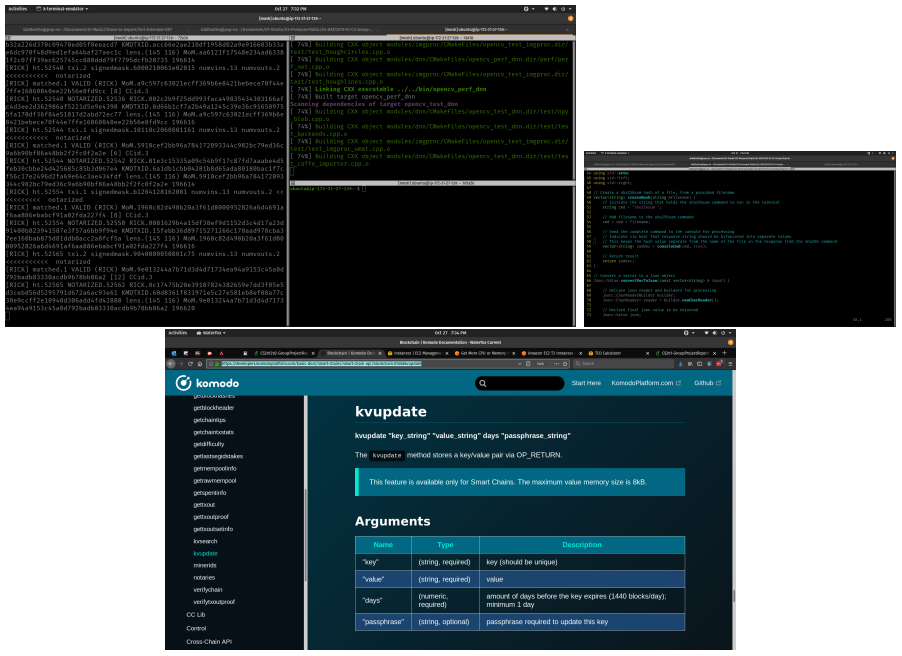

Figure 4: Bryan Beus Artifacts

Now that I am able to get a response from the blockchain, showing the location of the stored hash, I have technically finished all that I originally set out to do here. We can continue to improve the program, including fixing many of the above issues, as well as implementing a GUI to beautify the final product.

Included here are several artifacts show casing my progress on the project. The three-split terminal screen shows (on the left) the blockchain syncing, (on the upper right) the OpenCV

library building on my VPS, and (lower right corner) open for commands.

The screenshot that shows a full terminal screen is my vim terminal, which I use for regular coding. The file shown is `notarization.cpp`.

The screenshot of the documentation shows the official documentation for the Komodo project, including the key-value storage API command used in the project.

4 Results

4.1 Henryk Sosnowski

My focus on this project was to handle the camera operation and image saving portion of the program, and both processes are operated using the main module of the OpenCV library. First a camera object is created, and an error check is performed in case the computer does not have a camera or the program is denied access, which will edit a referenced value to halt the program. Next an empty window is created to give the user the control they expect from a basic window. Then a video window is created with the same name which will combine them both into a live camera feed that can be dragged and closed with proper functionality. Once any key is pressed, the current frame is saved to a file with a standard name, the camera feed closes, and the program automatically continues with Bryan's code to create a hash of the newly taken image. As both developers of this software were working on different operating systems (Windows and Unix), several concessions had to be made in function use and overall program design. The video capture function of OpenCV allowed us to call the computer's default camera settings, so in this way the camera can be utilized using the same code regardless of the user's platform. Our decision to save all files only in the source directory also allows every function that saves or references a file to not need to worry about the differences in operating system directory structure. While the base functionality works on both operating systems, our chosen method to gener-

ate a hash and connect to a blockchain was designed to run on Unix based systems, so native Windows support ultimately had to be dropped, but it can still be run on a Unix shell.

4.2 Bryan Beus

Our software successfully accomplishes the basic tasks we originally set out to achieve. The software captures an image from the user's webcam, creates a hash using `sha256sum` software, and stores this hash in a blockchain for future retrieval. This was the minimum viable product we intended to build. The software requires that the user have the Komodo `RICK` blockchain installed and synced.

This software relies on several libraries and resources. We use the OpenCV library to capture image data from the user's webcam. We use a custom-made `consoleCmd` function that relies on the `popen()` method to send raw commands at the Unix terminal. With this method, we send a raw request to the `sha256sum` preinstalled GNU/Linux software to obtain a hash of the captured image data. We used the `<jsoncpp>` library and the `consoleCmd` to send a raw `curl` command to the Komodo software daemon, and obtain returned json data. This is where the blockchainification occurs. We make frequent use of streams, vectors, and strings, as taught in our CS 201 class, to ensure that data is properly transferred throughout the program. All relevant information is printed into the console, allowing the user to observe the process as it proceeds.

5 Post Mortem

5.1 Henryk Sosnowski

Opening the camera and saving an image using OpenCV went well and did not cause many problems on both our operating systems. Passing the file between our two blocks of code also was simple as we simply kept the name and directory standard. Com-

munication between myself and my partner was also much easier than I was expecting through GitHUB and discord. Working out problems between our separately developed code sections was no trouble at all.

The obvious thing that went wrong is the dropping of windows support due to our chosen method of generating a hash. The next related thing was the loss of our GUI, which had already been coded and was working with the camera software. As it could no longer run on Windows, I was unable to continue development. We decided it was a superficial feature that was originally intended to be a bonus if everything else was running smoothly. I had a potential bonus idea to create a database of past images taken and their hashes, but after a bit of research the implementation was outside of my expertise and had to be dropped.

5.2 Bryan Beus

Three things that went well include the following. The software works as expected; facehasher successfully creates and stores a hash of a user's visual data, keeping it permanently in a blockchain. Furthermore, we were able to test blockchain software in a development environment, which has been a desire of my own for several years now, and we used C++ to do so. Finally, we were able to practice many of the programming principles taught in class, increasing our knowledge of the C++ language.

On the other hand, three things that "went wrong" include the following. Cross-platform compatibility ultimately was not feasible at our skill level. We tried on multiple occasions, each time lasting longer than four hours, to create cross-platform compatible code, but there are too many nuances. In the future, we may increase our dependence on libraries to solve this problem. We also were not able to create a GUI. This was not an issue regarding skill level, but rather time constraints. Nothing else "went wrong", per se, but we do hope to improve our coding abilities in the future.

6 Sample Output

6.1 Artifact 1

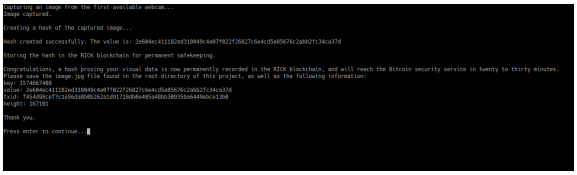

Figure 5: Console Readout for the Facehasher Software

6.2 Artifact 2

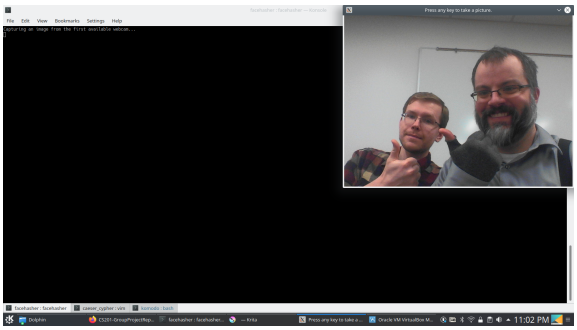

Figure 6: OpenCV Library In Action

6.3 Artifact 3

```
Capturing an image from the first available webcam...
Image captured.

Creating a hash of the captured image...

Hash created successfully. The value is:
    2e604ec411182ed318049c4a07f022f26827c6e4cd5a85676c2abb2fc34ca37d

Storing the hash in the RICK blockchain for permanent safekeeping.
```

```
9
10 Congratulations, a hash proving your visual data is now
   ↪  permanently recorded in the RICK blockchain, and will reach
   ↪  the Bitcoin security service in twenty to thirty minutes.
11 Please save the image.jpg file found in the root directory of
   ↪  this project, as well as the following information:
12 key: 1574667408
13 value:
   ↪  2e604ec411182ed318049c4a07f022f26827c6e4cd5a85676c2abb2fc34ca37d
14 txid:
   ↪  f454d98cef7c1e56da8b0b262a1d91718db0e405a48bb30935ba6449ebce13b0
15 height: 167101
16
17 Thank you.
18
19 Press enter to continue...
```

7 Program Source Code

The source code is licensed under the MIT license.

7.1 facehasher.hpp

```
1  /**
2   * facehasher.hpp
3   * CS 201
4   * Henryk Sosnowski and Bryan Beus
5   * October 12, 2019
6   *
7   */
8
9  #ifndef _FACEHASHER_HPP_
10 #define _FACEHASHER_HPP_
11
12 #include <iostream>
13 #include <string>
14 #include <vector>
15 #include <algorithm>
16 #include <sstream>
17 #include <fstream>
18 #include <stdio.h>
19 #include <stdlib.h>
20 #include <chrono>
21 #include <thread>
22 #include <functional>
23 #include <jsoncpp/json/json.h>
```

```
24 #include <jsoncpp/json/reader.h>
25 #include <jsoncpp/json/writer.h>
26 #include <jsoncpp/json/value.h>
27 #include <opencv2/opencv.hpp>
28
29 using std::cout;
30 using std::cin;
31 using std::endl;
32 using std::vector;
33 using std::string;
34 using std::noskipws;
35 using std::getline;
36 using std::istringstream;
37 using std::ostringstream;
38 using std::find;
39 using std::ifstream;
40 using std::function;
41 using std::setw;
42 using std::left;
43 using std::right;
44
45 // Send a command to the console
46 vector<string> consoleCmd(const string &input, bool bifurcate);
47
48 // Create a sha256sum hash of a file
49 vector<string> createHash(string &filename, bool& canContinue);
50
51 // Create a KV store transaction and process it on the blockchain
52 vector<string> createKVStore(const vector<string> & sumVec, bool&
   ↪  canContinue);
53
54 // Request a json response from a software daemon
55 Json::Value convertVecToJson(const vector<string> & input);
56
57 // Capture an image from a webcam
58 void image_capture(bool & canContinue);
59
60 #endif
```

7.2 facehasher_main.cpp

```
1 /**
2  * facehasher_main.cpp
3  * CS 201
4  * Henryk Sosnowski and Bryan Beus
5  * October 12, 2019
6  * Outputs data for facehasher into a GUI display
7  */
8
9 #include <iostream>
```

```cpp
#include <string>
#include <vector>
#include <algorithm>
#include <sstream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <chrono>
#include <thread>
#include <functional>
#include <jsoncpp/json/json.h>
#include <jsoncpp/json/reader.h>
#include <jsoncpp/json/writer.h>
#include <jsoncpp/json/value.h>

#include "facehasher.hpp"

using std::cout;
using std::cin;
using std::endl;
using std::vector;
using std::string;
using std::noskipws;
using std::getline;
using std::istringstream;
using std::ostringstream;
using std::find;
using std::ifstream;
using std::function;
using std::setw;
using std::left;
using std::right;

// Clear the console
void clearConsole();

// Wait for user response
void waitForContinue();

int main(int argv, char **argc) {

  // Initiate bool to track if each stage of the program passes successfully, and to cease program if any section fails
  bool canContinue = true;

  clearConsole();

  cout << "Capturing an image from the first available webcam..." << endl;

  // Capture an image from a webcam
  image_capture(canContinue);

```

```
61  // Test if image capture performed successfully
62  if (!canContinue)
63    return 0;
64
65  // Begin processing hash
66  cout << endl << "Creating a hash of the captured image..." <<
    ↪  endl;
67
68  // Choose file name that is hashed by default
69  string filename = "image.jpg";
70
71  // Create vector to hold file hash
72  vector<string> sumVec = createHash(filename, canContinue);
73
74  // Test if hash performed successfully
75  if (!canContinue)
76    return 0;
77
78  // Print hash to console
79  cout << endl << "Hash created successfully. The value is: " <<
    ↪  sumVec[0] << endl;
80  cout << endl << "Storing the hash in the RICK blockchain for
    ↪  permanent safekeeping." << endl;
81
82  // Obtain the kvStore response as a string
83  vector<string> kvStore = createKVStore(sumVec, canContinue);
84
85  // Test if createKVStore performed successfully
86  if (!canContinue) {
87    cout << "CreateKVStore func did not perform as expected" <<
      ↪  endl;
88    return 0;
89  }
90
91  // Convert the kvStore value to a Json::Value object for parsing
92  Json::Value jsonKVStore = convertVecToJson(kvStore);
93
94  // Test that json values were properly converted, and if not,
    ↪  end program
95  if (jsonKVStore.size() < 3) {
96    cout << endl << "jsonKVStore value is too small" << endl;
97    return 0;
98  }
99
100 // Print json values to console
101
102 cout << endl;
103 cout << "Congratulations, a hash proving your visual data is
    ↪  now permanently recorded in the RICK blockchain, and will
    ↪  reach the Bitcoin security service in twenty to thirty
    ↪  minutes." << endl;
```

```
104   cout << "Please save the image.jpg file found in the root
      ↪  directory of this project, as well as the following
      ↪  information: " << endl;
105   cout << "key: " << jsonKVStore["result"].get("key", "default
      ↪  value").asString() << endl;
106   cout << "value: " << jsonKVStore["result"].get("value",
      ↪  "default value").asString() << endl;
107   cout << "txid: " << jsonKVStore["result"].get("txid", "default
      ↪  value").asString() << endl;
108   cout << "height: " << jsonKVStore["result"].get("height",
      ↪  "default value").asString() << endl;
109
110   cout << endl << "Thank you." << endl;
111
112   waitForContinue();
113
114   return 0;
115
116 }
117
118 // Clear the console
119 void clearConsole() {
120
121     // Clear the console
122     cout << "\033[2J\033[1;1H";
123 }
124
125 // Wait for user response
126 void waitForContinue() {
127     cout << endl << "Press enter to continue...";
128     getchar();
129 }
```

7.3 image_capture.cpp

```
1  //image_capture.cpp
2  //CS 201
3  //Henryk Sosnowski and Bryan Beus
4  //October 12, 2019
5   //Captures an image from a web camera and saves the image to a
     ↪  local folder
6
7
8  #include <iostream>
9  #include <opencv2/opencv.hpp>
10 #include "facehasher.hpp"
11
12
```

```cpp
//This function opens a new window with the computer's default camera settings and allows the user to save the current frame as a jpg.
//It returns true if an image was saved or false if it failed to open the camera.
void image_capture(bool & canContinue)
{
  //Attempt to open the camera using the computer's default camera settings
  cv::VideoCapture cap(0);

  //Exits the function if there is an issue accessing the camera
  //Changes the referenced bool value to signal the program to stop
  if (!cap.isOpened())
  {
    std::cout << "Error: Cannot open camera.";
    canContinue = false;
    return;
  }

  //The window name visible to the user
  std::string windowName = "Press any key to take a picture.";

  //This Mat object will story the current frame
  cv::Mat frame;

  //Opens a currently empty window with the passed name
  cv::namedWindow(windowName, cv::WINDOW_AUTOSIZE);
  while (true)
  {
    //Opens a camera object with live feed visible to the user.
    //The name must be the same as the previous window for them to combine for complete functionality
    cap >> frame;
    cv::imshow(windowName, frame);

    //The program waits until a key press is detected
    char key = cv::waitKey(30);

    //Any key press will save the current frame to the source directory as "image.jpg"
    if (key > 0)
    {
      cv::imwrite("image.jpg", frame);
      std::cout << "Image captured." << std::endl;
      break;
    }
  }

  //The window is destroyed upon completion.  Necessary for UI integration.
```

```
  cv::destroyWindow(windowName);

  return;
}
```

7.4 notarization.cpp

```
/**
 * notarization.cpp
 * CS 201
 * Henryk Sosnowski and Bryan Beus
 * October 12, 2019
 * Hashes the image and inserts the hash into a notarized
 ↪  blockchain
 */

#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <sstream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <chrono>
#include <thread>
#include <functional>
#include <jsoncpp/json/json.h>
#include <jsoncpp/json/reader.h>
#include <jsoncpp/json/writer.h>
#include <jsoncpp/json/value.h>

#include "facehasher.hpp"

using std::cout;
using std::cin;
using std::endl;
using std::vector;
using std::string;
using std::noskipws;
using std::getline;
using std::istringstream;
using std::ostringstream;
using std::find;
using std::ifstream;
using std::function;
using std::setw;
using std::left;
using std::right;
```

```
42
43 // Create a sha256sum hash of a file, from a provided filename
44 vector<string> createHash(string &filename, bool& canContinue) {
45     // Initiate the string that holds the sha256sum command to
       ↪  run in the terminal
46
47 #ifdef _WIN32
48
49     string cmd = "sha256sum.exe ";
50
51 #elif __unix__
52
53   string cmd = "sha256sum ";
54
55 #endif
56
57     // Add filename to the sha256sum command
58     cmd = cmd + filename;
59
60     // Send the complete command to the console for processing
61     // Indicate via bool that response string should be
       ↪  bifurcated into separate values
62     // This keeps the hash value separate from the name of the
       ↪  file in the response from the sha256 command
63     vector<string> sumVec = consoleCmd(cmd, true);
64
65     if (sumVec.size() < 1) {
66     cout << "Error in creating hash. Ensure that sha256sum
       ↪  software is running properly on your machine" << endl;
67     canContinue = false;
68     }
69
70     // Return result
71     return sumVec;
72 }
73
74 // Convert a vector to a json object
75 Json::Value convertVecToJson(const vector<string> & input) {
76
77     // Delcare json reader and builders for processing
78     Json::CharReaderBuilder builder;
79     Json::CharReader* reader = builder.newCharReader();
80
81     // Declare final json value to be returned
82     Json::Value json;
83
84     // Declare string to hold errors
85     string errors;
86
87     // Declare bool variable to test if parse is succesful
88     // Pass into parse() function the first element in the input
       ↪  vector
89     // This element holds the result from the call to the
       ↪  blockchain
```

```
90      // Also pass in the json variable and errors variable
91      bool parsingSuccessful = reader -> parse(
92              input[0].c_str(),
93              input[0].c_str() + input[0].size(),
94              &json,
95              &errors
96      );
97
98      // Clean up pointer
99      delete reader;
100
101     // Test if parsing was successful, and if not, print to
        ↪  console
102     if (!parsingSuccessful) {
103         cout << "Failed to parse the JSON, errors: " << endl;
104         cout << errors << endl;
105     }
106
107     // Return json values
108     // Or empty vector
109     return json;
110 }
111
112 // Send a hash to the RICK blockchain for decentralized storage
113 vector<string> createKVStore(const vector<string> & sumVec, bool&
    ↪  canContinue) {
114
115     // Create a timestamp in string form
116     // This acts as the key value
117     time_t t = std::time(0);
118     string key = std::to_string(t);
119
120     // Create the full curl command to execute in the terminal
121     string updateKey = "curl --silent --user
        user3789197396:passe6ba30ce2b58b085b63f739613aee3016766c6dc6daaf9e74b1a89adf
        ↪  --data-binary '{\"jsonrpc\": \"1.0\",
        ↪  \"id\":\"curltest\", \"method\": \"kvupdate\",
        ↪  \"params\": [\"" + key + "\", \"" + sumVec[0] + "\",
        ↪  \"2\"] }' -H 'content-type: text/plain;'
        ↪  http://127.0.0.1:25435";
122
123     // Send string command to consoleCmd() function to process
124     // Store returned vector as sendKey
125     vector<string> sendKey = consoleCmd(updateKey, false);
126
127     if (sendKey.size() < 1) {
128       canContinue = false;
129       cout << "Error creating KV store transaction. Ensure the
          ↪  RICK asset chain is launched and synced." << endl;
130     }
131
132     return sendKey;
```

```
133 }
134
135 // Execute a command in the Unix terminal, from a provided string
136 // If necessary, cut the returned string response into separate
    ↪  elements in the to-be-returned vector
137 vector<string> consoleCmd(const string &input, bool bifurcate) {
138
139     // Reference to this tutorial
140     //
        ↪  https://www.jeremymorgan.com/tutorials/c-programming/how-to-capture-the-outp
141
142     // Create string to hold command after adjustments
143     string cmd = input;
144
145     // Create string to hold returned data
146     string data;
147
148     // Create file stream and pointer
149     FILE * stream;
150
151     // Declare max buffer size
152     const int max_buffer = 256;
153
154     // Create buffer array of char type
155     char buffer[max_buffer];
156
157     // If bifurcate == true, instruct the console to store all
        ↪  values in the "1" variable in bash terminal
158     if (bifurcate)
159         cmd.append(" 2>&1");
160
161     //
162 #ifdef _WIN32
163
164     // Execute the command as a c string, using popen()
165     stream = _popen(cmd.c_str(), "r");
166
167 #elif __unix__
168
169     // Execute the command as a c string, using popen()
170     stream = popen(cmd.c_str(), "r");
171
172 #endif
173     // If there is a positive returned response
174     // Push this response into the data variable
175     if (stream) {
176         while (!feof(stream)) {
177             if (fgets(buffer, max_buffer, stream) != NULL)
178                 data.append(buffer);
179         }
180
181 #ifdef _WIN32
182         // End the shell process
```

```
183         _pclose(stream);
184     }
185
186 #elif __unix__
187
188         // End the shell process
189         pclose(stream);
190     }
191
192 #endif
193
194     // Declare a stream and a vector to hold final values
195     istringstream instream(data);
196     vector<string> res;
197
198     // If bifurcating result, use the stream to separate each
        ↪  input and push into res
199     if (bifurcate) {
200         while (!instream.eof()) {
201             string temp;
202             instream >> temp;
203             res.push_back(temp);
204         }
205
206     // Otherwise, push the entire response into the first element
        ↪  of the res vector
207     } else {
208         res.push_back(data);
209     }
210
211
212     return res;
213 }
```

References

[Fac19] Factom Inc. Proof of process. https://www.factom.com/solutions/integrate/, 2019. [Online; accessed 6-October-2019].

[Lis19] Lisa Eadicicco. There's a terrifying trend on the internet that could be used to ruin your reputation, and no one knows how to stop it. https://www.insider.com/dangerous-deepfake-technology-spreading-cannot-be-stopped-2019-7, 2019. [Online; accessed 6-October-2019].