CS 201 Group Project
Face Hasher

Henryk Sosnowski
Bryan Beus

October 28, 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

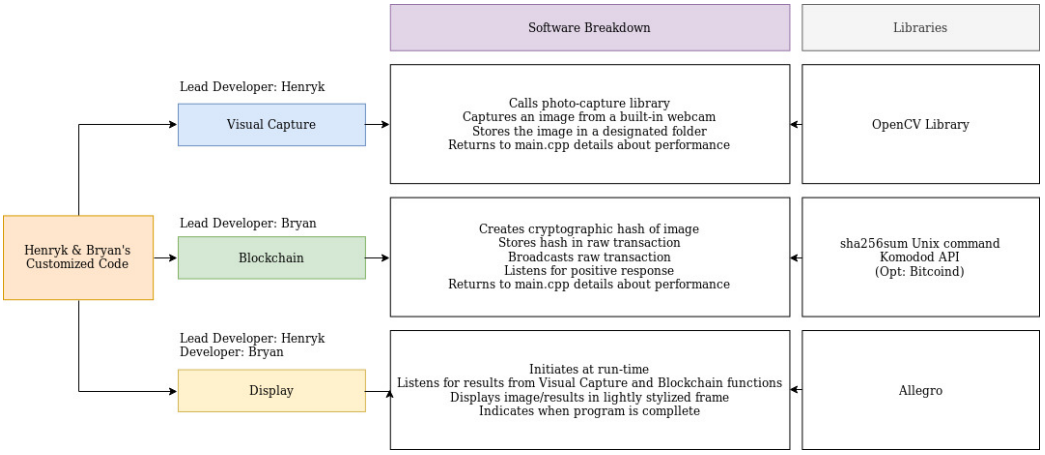

Figure 1: Software Architecture

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.

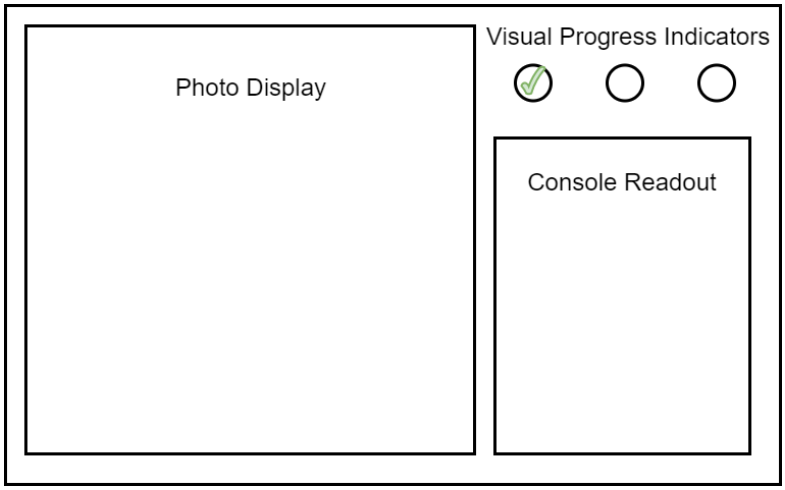

Figure 2: User Interface Design

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.

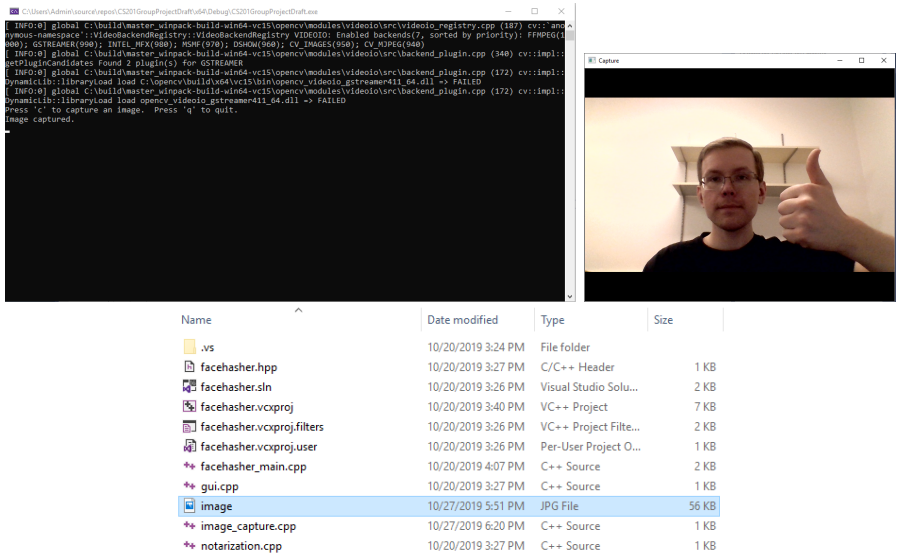

Figure 3: Henryk Sosnowski Artifacts

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.

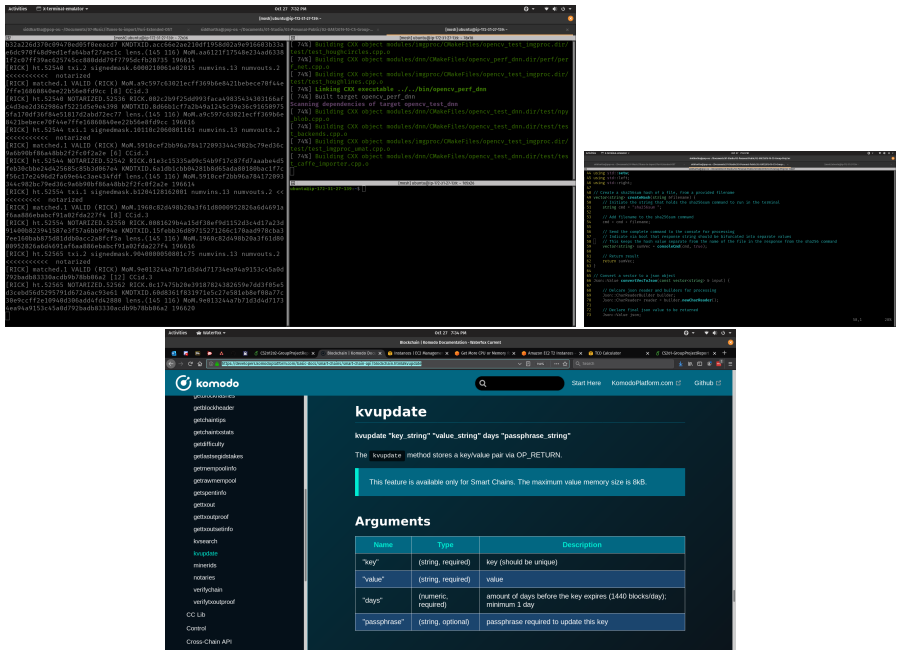

Figure 4: Bryan Beus Artifacts

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.

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].