

Mirror Key Reflection-Based Biometric Validation System with Forensic Logging

Alagu Murugan K¹, Magna V²,

¹Alagu Murugan. K Department of Cyber Forensics & Dr. M.G.R. Educational and Research Institute

²Magna. V Department of Cyber Forensics & Dr. M.G.R. Educational and Research Institute

Abstract - The old way of using passwords to get into systems is still not safe. It can be broken into using force people can steal passwords and bad people can trick others into giving away their passwords. This paper is about a way to get into systems using a Mirror Key Reflection-Based Biometric Validation System. It is a system that uses the web and recognizes faces to let people in. We used Python and other tools like Django and OpenCV to build this system.

The system takes a picture of the users face using a webcam and then compares it to a special code that is stored in a database. This code is very safe because it is encrypted. The system decides if the user is real or not in time, by comparing the picture to the stored code. Every time someone tries to get into the system it makes a record of what happened. This record is very safe because it uses hashing that can tell if someone tried to change it.

We tried out the system. It worked very well when the lighting was good. It could tell who people were. Did not let in anyone who was not supposed to be there. The system is also easy to add things to, like checking if the person is real or just a picture. This is important because it means we can use this system in schools and small businesses without having to buy special equipment. The Mirror Key Reflection-Based Biometric Validation System is a way to get into systems without using passwords. It uses face recognition. Is very safe.

Key Words: Face recognition, biometric authentication, AES encryption, forensic logging, OpenCV, Django, SHA-256, deep learning embeddings

1.INTRODUCTION

The first thing we need to think about when we talk about information systems is authentication. This is the line of defence. We all know that using passwords and other knowledge-based credentials has a lot of limitations. These limitations have been talked about a lot in security

literature. The problem with passwords is that they can be forgotten, shared or stolen. When we use multi-factor schemes with possession tokens it can be a hassle to manage them.

Biometric modalities, which are like behavioural traits that are unique to each person have been getting a lot of attention from researchers. This is because these traits cannot be forgotten or easily transferred.

Among all the modalities face recognition is special. It is not intrusive. We only need a standard camera. It is also natural for humans to identify each other by their faces. The technology for face recognition has improved a lot with learning-based face descriptors. This has made it possible to do real-time face authentication on computers.

With all this progress it is still not common to see face recognition systems in places like academic labs or small offices. Many systems need sensors or special software that can be slow and raise privacy concerns. That is why we need a system that's open-source and can be used for face recognition with secure storage and logging.

This paper is about the Mirror Key Reflection Biometric Validation System. The name of the system is based on the idea that our face's like a mirror key that unlocks our identity. The system is built using open-source components and can run on a regular computer. The main things the system can do are:

- * It has a biometric authentication pipeline from capturing a face on a webcam to making a decision about access.

- * It uses AES-256 encryption to protect the templates so even if someone gets into the database they cannot get the raw biometric data.

- * It has a logging module that uses SHA-256 hashing, which helps with investigating incidents and keeping track of what happens.

- * It has a web application architecture that makes it easy to add new things in the future.

The rest of this paper will be about the following things. Section 2 will talk about related work. Section 3 will describe the system architecture and design. Section 4 will go into details, about how the system was implemented. Section 5 will show the results of experiments. Section 6 will discuss what can be done to improve the system in the future. Section 7 will conclude the paper.

2.RELATED WORKS

The first thing we need to think about when we talk about information systems is authentication. This is the line of defence. We all know that using passwords and other knowledge-based credentials has a lot of limitations. These limitations have been talked about a lot in security literature. The problem with passwords is that they can be forgotten, shared or stolen. When we use multi-factor schemes with possession tokens it can be a hassle to manage them.

Biometric modalities, which are like behavioural traits that are unique to each person have been getting a lot of attention from researchers. This is because these traits cannot be forgotten or easily transferred. Among all the modalities face recognition is special. It is not intrusive. We only need a standard camera. It is also natural for humans to identify each other by their faces. The technology for face recognition has improved a lot with learning-based face descriptors. This has made it possible to do real-time face authentication on computers.

With all this progress it is still not common to see face recognition systems in places like academic labs or small offices. Many systems need sensors or special software that can be slow and raise privacy concerns. That is why we need a system that's open-source and can be used for face recognition with secure storage and logging. This paper is about the Mirror Key Reflection Biometric Validation System. The name of the system is based on the idea that our face's like a mirror key that unlocks our identity. The system is built using open-source components and can run on a regular computer. The main things the system can do are:

- * It has a biometric authentication pipeline from capturing a face on a webcam to making a decision about access.
- * It uses AES-256 encryption to protect the templates so even if someone gets into the database they cannot get the raw biometric data.

* It has a logging module that uses SHA-256 hashing, which helps with investigating incidents and keeping track of what happens.

* It has a web application architecture that makes it easy to add new things in the future.

The rest of this paper will be about the following things. Section 2 will talk about related work. Section 3 will describe the system architecture and design. Section 4 will go into details, about how the system was implemented. Section 5 will show the results of experiments. Section 6 will discuss what can be done to improve the system in the future. Section 7 will conclude the paper.

3. SYSTEM ARCHITECTURE

3.1 Architectural Overview

The MKR system is made up of layers. It has a part that people can see and use in their browser. This part talks to the Django backend using the internet. The backend is like a manager. It is, in charge of the MKR system. The MKR system has four parts: the biometric pipeline, the encryption module, the database layer and the forensic logger. These parts all work together. Figure 1 shows how the data moves through the MKR system.

Module	Responsibility
Frontend (HTML/JS/CSS)	Renders webcam feed, captures frames, submits credentials via AJAX, displays result
Django Backend	Processes HTTP requests, coordinates module calls, enforces authentication logic
OpenCV Module	Detects face region in captured frame, normalises image for embedding extraction
face_recognition	Computes 128-dimensional facial embedding using dlib deep residual network
AES-256 Encryption	Encrypts template before storage; decrypts template during comparison
MySQL Database	Persists encrypted templates (UserProfile) and authentication events (ForensicLog)

Forensic Logger	Records timestamp, match score, decision, IP address, and SHA-256 event hash
-----------------	--

Table 1: MKR System Module Summary

3.2 Data Flow During Authentication

When someone tries to log in the browser uses JavaScript to take a picture from the webcam. It turns this picture into a special kind of string. This string is sent to the Django backend using a kind of message. The backend then looks at the picture finds the face in it using OpenCV and cuts out the face part. It then uses the face recognition library to get information from the face. This information is like a set of numbers that is 128 numbers long. The system then compares this set of numbers to the one it already has. If the two sets of numbers are close enough like within a distance it means the person is who they say they are. The system uses a number called a threshold to decide how close is close enough. If the distance is below this threshold, which is usually set to 0.5 the system says yes this is the person. The whole thing that happened including how sure the system's the decision it made is then written down in a special log along with the persons internet address and a special code before the system sends a response back, to the browser.

3.3 Template Security Model

The biometric information that is stored in a database is a problem. If someone bad gets into the database they can get all the information and that is very bad because we can not change our biometric information. The MKR system tries to fix this by using a strong encryption called AES-256. When someone new joins, a secret key is made just for them. This key is used to lock up their information so it is safe. The locked up information and the secret key are both stored in the database.

This way even if someone gets into the database they will only get the locked up information and not the real biometric information. It is like having a safe and the key to the safe is also locked up. The MKR system does not try to separate the key from the locked up information, which's not perfect.. It does make it harder for someone bad to get the biometric information because they would need to know a lot, about the database and the application to get it.

4. IMPLEMENTATION

4.1 Technology Stack

The system is made using Python 3.10 on a Windows computer. We use Django 4.2 for the website because it

has a way of working with databases a built-in admin page and a simple way to set up projects. The database is MySQL 8.0. We use the mysqlclient adapter to access it. When the system needs to look at pictures and videos it uses OpenCV 4.8. To find faces in pictures it uses face_recognition 1.3.0, which gets help from dlib and its special face finder. The system also uses AES encryption, which is provided by the PyCryptodome library. All the parts we need are easy to install using pip and a list of requirements. The total size of the installation is 2.5 GB, which is mostly because of dlib and the things it needs to work. The system is implemented in Python 3.10. The web framework is Django 4.2. The database backend is MySQL 8.0. Computer vision operations use OpenCV 4.8. Face embedding extraction uses face_recognition 1.3.0.

4.2 Biometric Pipeline

Face detection uses OpenCVs Haar Cascade classifier to find the face. Then it uses face_recognitions HOG-based detector to align the face features. The face_recognition.face_encodings() function gives a list of 128-element arrays one for each face found.

* For enrolment if than one face is detected the system rejects the frame. It asks the user to make sure they are alone in the frame.

For authentication it takes the face found and compares it to the stored template. It uses face_recognition.compare_faces() and face_recognition.face_distance() to compare. This comparison calculates the difference, between the descriptor vectors.

4.3 Encryption and Decryption

Template encryption is a process that involves steps. First the NumPy embedding array is turned into bytes using numpy.tobytes(). Then a special key is made. This key is 32 bytes long, which is 256 bits. It is created with os.urandom().Next a 16 byte thing called an initialisation vector or IV for short is added to the beginning of the output. After that the encryption happens. This is done using PyCryptodomes AES.new(). It is in CBC mode, with something called PKCS7 padding.

The result of all this is a byte string that gets stored in the database BinaryField. To decrypt this the process is reversed. The IV is taken from the 16 bytes. Then the AES-CBC decryption is applied. The padding that was added is removed. Finally numpy.frombuffer() is used to put the embedding array together so it can be compared

to the original Template encryption array. This way the Template encryption array is safe and secure.

4.4 Forensic Logging Module

When someone tries to log in it does not matter if they are allowed in or not the forensic logger is always called. The logger takes the username, the time of the attempt how good the match was, what the decision was and the ip address of the computer that tried to log in. It puts all of these things together into one string. This string is then run through a kind of math called SHA-256, which is done using a python tool and the result is stored with all the other information in the ForensicLog part of the database. If someone were to try to change a log after it has been made the math would not add up anymore. This would be easy to see when someone checks the logs. The people who are in charge can use a tool to look at the logs but they can not change them so they can see what has been happening with log ins without having to look directly at the database. The forensic logger is very important for the ForensicLog and the ForensicLog is very important, for keeping track of the authentication attempts.

4.5 Web Interface

The frontend has three HTML pages. These pages are served by Django's template engine.

* A registration page

* A login page

* A forensic dashboard

The registration and login pages have a webcam feed. This feed works through the browsers MediaDevices API. A JavaScript function takes a video frame. It draws the frame on a HTML5 canvas. Then it turns the frame into a Base64 JPEG string. This string is in a hidden form field when you submit the form. The forensic dashboard shows the ForensicLog table. The rows in the table have colours.

* Green for ACCEPT

* Red for REJECT

It also shows a version of the SHA-256 hash for each record. No extra JavaScript libraries are used. All interactive parts work with JavaScript. The forensic

dashboard displays ForensicLog table data, with colour codes and truncated SHA-256 hashes. The dashboard uses vanilla JavaScript for interactivity.

5. EXPERIMENTAL EVALUATION

5.1 Experimental Setup

We did some testing, on a laptop that had an Intel Core 10th generation processor and 8 GB of memory. This laptop also had a built-in 720p webcam. The MySQL server was running on this laptop. We got ten people to help us with the testing. Each of these people tried to register three times. We kept one of these attempts as a sample. Then each person tried to log in ten times under two lighting conditions. One was the light you find indoors and the other was a room that was not very well lit. We also tried to trick the system ten times for each person by showing a printed picture of their face on another monitor. We made sure that all of these tests were done with the default setting of 0.5.

5.2 Results

Condition	TAR(%)	FAR(%)	FRR(%)
Normal indoor lighting	94	0	6
Dim lighting	81	0	19
Photo spoof attempt	—	0	—

Table 2: Authentication Performance Summary (Threshold = 0.5)

TAR: True Acceptance Rate; FAR: False Acceptance Rate; FRR: False Rejection Rate. The dash (—) indicates that TAR and FRR metrics are not applicable for spoofing attempts, which are assessed solely by FAR.

The system works well with normal indoor lighting. It can recognize people about 94 percent of the time. When it does not recognize someone it is usually because the face is not fully in the picture. The system does not let in anyone who should not be let in. When the lighting is not so good the system has a time recognizing faces. 19 Percent of the time it will not recognize a face when the lighting is dim. This is because the system has trouble finding the face in the picture.

The good thing is that the system can tell when someone is trying to trick it with a photo. It knows the difference between a face and a picture of a face. This is because the system is sensitive to the shape of a face.

It takes the system about 380 milliseconds to look at a picture and decide if the person is who they say they are. This is fast enough, for using the system to log in to a computer.

5.3 Forensic Log Verification

To check if the integrity mechanism is working properly we manually got 20 ForensicLog records from the MySQL database. We then recalculated their SHA-256 hashes using the stored field values. All 20 records had matching hashes, which means no changes were made without being detected during the test.

We also did a test where we manually changed the decision field of three records from REJECT to ACCEPT using a SQL UPDATE statement. In all three cases the recalculated hash was different, from the stored hash, which shows that the system can effectively detect changes made to the records after they were created. The integrity mechanism is working as expected. The ForensicLog records and their SHA-256 hashes are valid.

6. DISCUSSION AND FUTURE ENHANCEMENTS

To check if the integrity mechanism is working properly we manually got 20 ForensicLog records from the MySQL database. We then recalculated their SHA-256 hashes using the stored field values. All 20 records had matching hashes, which means no changes were made without being detected during the test.

We also did a test where we manually changed the decision field of three records from REJECT to ACCEPT using a SQL UPDATE statement. In all three cases the recalculated hash was different, from the stored hash, which shows that the system can effectively detect changes made to the records after they were created. The integrity mechanism is working as expected. The Forensic Log records and their SHA-256 hashes are valid.

7. CONCLUSIONS

This paper is about the Mirror Key Reflection Biometric Validation System. The Mirror Key Reflection Biometric Validation System is a web application that people can use for free. It is a face recognition system that helps to verify who a user is. The Mirror Key Reflection

Biometric Validation System stores information in a way using something called AES-256 encryption. It also keeps a record of everything that happens with SHA-256 logging.

The Mirror Key Reflection Biometric Validation System was built using Python and other tools like Django and OpenCV. The Mirror Key Reflection Biometric Validation System works on computers without needing special equipment or internet connections.

We tested the Mirror Key Reflection Biometric Validation System with ten people. The Mirror Key Reflection Biometric Validation System worked correctly for nine out of ten people when the lighting was normal. The Mirror Key Reflection Biometric Validation System did not make any mistakes when someone tried to trick it. The logging part of the Mirror Key Reflection Biometric Validation System was able to detect when someone tried to change the records on purpose.

These results show that the Mirror Key Reflection Biometric Validation System is more secure than using passwords. The Mirror Key Reflection Biometric Validation System is also easy to use for developers who're not experts in biometrics.

The Mirror Key Reflection Biometric Validation System is made in a way that makes it easy to add features. For example we can add a way to check if someone is really alive or a way to manage keys separately or a way to use different methods to verify who someone is. This makes the Mirror Key Reflection Biometric Validation System a good starting point for research and, for using it in real life especially in schools and small businesses.

ACKNOWLEDGEMENT

The authors would like to thank the volunteer participants who contributed their time for the experimental evaluation. The authors also acknowledge the open-source communities behind the OpenCV, dlib, face_recognition, Django, and PyCryptodome projects, without which this work would not have been feasible.

REFERENCES

- [1] R. Ramachandra and C. Busch, "Presentation Attack Detection Methods for Face Recognition Systems: A Comprehensive Survey," *ACM Computing Surveys*, vol. 50, no. 1, pp. 1–37, 2021.
- [2] M. Wang and W. Deng, "Deep Face Recognition: A Survey," *Neurocomputing*, vol. 429, pp. 215–244, 2021.
- [3] X. Wu, R. He, Z. Sun, and T. Tan, "A Light CNN for Deep Face Representation with Noisy Labels," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2884–2896, 2018. (Cited for contextual framing of classical vs. deep approaches.)
- [4] A. Adjabi, A. Ouahabi, A. Benzaoui, and A. Taleb-Ahmed, "Past, Present, and Future of Face Recognition: A Review," *Electronics*, vol. 9, no. 8, p. 1188, 2020.
- [5] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "ArcFace: Additive Angular Margin Loss for Deep Face Recognition," in *Proc. IEEE/CVF CVPR*, pp. 4690–4699, 2019.
- [6] C. Rathgeb and A. Uhl, "A Survey on Biometric Cryptosystems and Cancelable Biometrics," *EURASIP Journal on Information Security*, vol. 2011, Article ID 3, 2022.
- [7] A. Valjarevic and H. S. Venter, "A Harmonised Digital Forensic Process Model," in *Proc. 2012 Information Security for South Africa*, IEEE, 2012. (Foundational reference on forensic readiness — cited for conceptual alignment.)
- [8] T. de Freitas Pereira, J. Komulainen, A. Anjos, J. M. De Martino, A. Hadid, M. Pietikäinen, and S. Marcel, "Face Liveness Detection Using Dynamic Texture," *EURASIP Journal on Image and Video Processing*, vol. 2014, Article ID 2, 2022.