IoT-based Access Management Supported by AI and Blockchains

Schiller, Eryk ; Esati, Elfat ; Stiller, Burkhard

Abstract: This work specifies, implements, and evaluates access management based on face recognition. The system developed uses Internet-of-Things (IoT) for video surveillance, Artificial Intelligence (AI) for face recognition, and Blockchains (BC) for immutable permanent storage and provides excellent properties in terms of image quality, end-to-end delay, and energy efficiency.

IoT-based Access Management Supported by AI and Blockchains

Eryk Schiller, Elfat Esati, Burkhard Stiller
Communication Systems Group CSG, Department of Informatics IfI, University of Zürich UZH
Binzmühlestrasse 14, CH—8050 Zürich, Switzerland
Emails: [schiller|stiller]@ifi.uzh.ch, elfat.esati@uzh.ch

***Abstract*—This work specifies, implements, and evaluates access management based on face recognition. The system developed uses Internet-of-Things (IoT) for video surveillance, Artificial Intelligence (AI) for face recognition, and Blockchains (BC) for immutable permanent storage and provides excellent properties in terms of image quality, end-to-end delay, and energy efficiency.**

***Index Terms*—Management of Access Control, Video Surveillance, Blockchains, Internet-of-Things, Artificial Intelligence**

I. Introduction

Internet-of-Things (IoT) is transforming the world of things, which impacts many economic sectors, such as manufacturing, transportation, automotive, consumer goods, and healthcare [6]. Thanks to the advances in the integrated circuit design, IoT devices are now equipped with powerful processors of a new generation, handling processing load efficiently [2], [11]. This offers an opportunity to run complex tasks on IoT devices in a distributed fashion. However, IoT comes with many challenges or gaps that still need to be improved [9], such as the centralization of various IoT platforms, *e.g.,* Amazon Web Services (AWS)-IoT, security and privacy issues concerning communication protocols as well as vulnerability to various attacks related to poor maintenance of IoT infrastructures, *e.g.,* Mirai [16].

Blockchains (BC) [5], [30] offer an immutable storage of data records in a distributed ledger by cryptographic measures. BCs can help IoT infrastructures deal with centralization: when IoT infrastructures store and process data in a BC, this removes the single-point-of-failure present in currently available IoT platforms, such as AWS IoT [9], [18], [22], [23]. BCs bring significant advantages in terms of information provenance, non-repudiation, and authenticity (while every originator signs every record with its private key), thus, increasing the overall information security in the system [7].

Finally, Artificial Intelligence (AI) plays a significant role in providing accurate data analysis in real-time. Nevertheless, the design and development of an efficient data analysis tool using AI comes with challenges, such as centralization and transparency [26]. Therefore, integrating BCs with AI can produce a robust approach to resolve those issues. Often, AI is considered as a black box providing classifiers or predictors, lacking transparency. However, the transparency can be materialized by ordering AI decisions among many nodes in a given BC. This provides a precise, immutable track of AI decisions ordered in time, which can, *e.g.,* form the basis for managerial access control decisions. Therefore, the simultaneous application of IoT, BCs, and AI, shows a successful synergy transforming data acquisition, analysis, and storage [7], [19], [25].

Having a closer look at the research in these three domains mentioned above, one may expect many proposals and architectures. However, there exist so far no addressed use-cases, wherein those three technologies may complement each other. Therefore, the main goal of this paper is the design and development of a BC-enabled IoT Architecture coupled with AI providing an efficient implementation of a use-case, here an access management approach.

The remainder of this paper is organized as follows. Section II introduces the related work. While Section III provides new use-cases and specifies the architecture of the system, Section IV details its implementation and evaluates the performance of the system developed. Finally, Section V summarizes the work and adds an outlook on future work.

II. Related Work

This overview addresses most recent projects and research on the integration of AI, BC, and IoT.

A. Internet-of-Things

IoT is transforming the interaction with everyday things [6]. Practically almost anything can be equipped with a microcontroller, sensors, and actuators, thus, allowing things to monitor the surrounding environment at a certain level of intelligence. A thing able to react to changing conditions of the environment is referred to as a smart object, which are often also connected to the Internet allowing for harvesting information by a centralized storage (*e.g.,* AWS IoT) [9] and more sophisticated information processing at the fog, edge, or cloud level as well as actuating in a given environment.

Most IoT devices depend on microcontrollers [2], [11] or microprocessors of minimal processing power, making them much less power-hungry than *e.g.,* a smartphone. This design choice is based on the assumption that processing power comes with significant energy expenses. Therefore, less efficient computing is energy-efficient, runs with minimal power, conserves energy, and allows for a longer lifetime of an object on the single battery charge.

B. Internet-of-Things and Artificial Intelligence

Due to low processing power of IoT devices, there is a need for green communication in the IoT domain [3] as well as lightweight approaches in AI. As a result, different algorithmic approaches are taking place, and more efficient algorithms are being developed [28]. This gives rise to novel research domains, such as Tiny Machine Learning [12], Tiny Deep Learning [17], which are able to operate on constrained IoT devices. As an example, MIT researchers [4] have implemented a system called MCUNet, which has a high potential of bringing deep learning to low capacity devices like tiny microcontrollers for performing tasks like image, audio, or video recognition.

One of the most recent studies [28] worked in the direction of image processing and cloud offloading. Two approaches with deep learning were tested, *i.e., (i)* cloud offloading of deep learning platforms and *(ii)* migrating deep learning to IoT devices. The two approaches were tested and looked from the perspective of reduced energy efficiency and real time requirements of object recognition. In the first approach, they used Convolutional Neural Networks (CNN) on the cloud, while the device was responsible for taking images and forward them to the cloud. The results show that executing machine learning on an IoT device consumed more energy compared to cloud offloading. However, offloading AI to the cloud also comes with drawbacks which has lead to a latency starting from 2 seconds that goes up to 5 seconds, which is far higher compared to the execution of AI on the IoT device. This infers that the variability of response time makes it quite unreliable and not useful for real time AI image processing.

Esp Eye [11], equipped with Tensilica LX6 dual-core processor, is most probably the first microcontroller device that performs real-time face recognition. However, this does not mean that the exact face detection and recognition algorithms for regular computers have been used here. Esp Eye is one of its kind that comes with Esp Who [27] platform, which supports both face detection and face recognition. To our knowledge, this is the only device that can perform video streaming combined with real-time face recognition in a microprocessor that lies outside of the main three classes of CPUs such as Intel, AMD, or ARM processors. Esp Who platform implements a framework called MTMN. MTMN refers to both MT-CNN (Multi-Task Cascaded Convolutional Networks) [32] and MobileNets (MN) [14] as well as Face Recognition model based on Convolutional Neural Network (FRMN) [24]. There are several deep learning techniques, which have been specified and implemented that paved the way towards face detection. However, MTCNN is a framework that integrates both face detection and alignment. With the help of MobileNets, it builds lightweight deep neural networks, which use depth-wise separable convolutions for face detection. The FRMN is a Convolutional Neural Network mixing local and global image features. It provides good feature extraction from the mouth, nose, and eyes, delivering enhanced and accurate face recognition.

C. Internet-of-Things and Blockchains

Narrowing down the scope to BC and IoT systems, one sees research attempts to close the gaps of IoT systems by removing the centralized control as well as attack the problem of provenance, non-repudiation, and authenticity in IoT data streams with the help of BC [9], [18], [22], [23]. Other studies [31] attempt to close security and privacy IoT gaps with the help of the BC to ensure the reliability and availability of the data.

D. Artificial Intelligence and Blockchains

There is a high research interest in BC and AI analyzed in various domains and applications. Some research [10] focuses on the application of AI in the BC for making BCs more efficient. It concentrates on BC consensus mechanisms and better governance. There are also research items about the applications of BC in AI. Like IoT, the AI domain also suffers from security, centralized architecture, and resource limitations. This is what BC promises to solve. There is a lot of discussion and research in this area. However, most of them are reviews and solutions that do not develop use-cases or provide actual implementations.

E. Artificial Intelligence, Blockchains, and Internet-of-Things

Several attempts shed the light on the benefits of converging IoT, BC, and AI [7]. However, most of them are either reviews or explorations but lack a concrete implementation in a use-case [19], [25]. More comparative research outcome in this domain is a so-called BlockIoTIntelligence attempt [26], which proposes an architecture that utilizes BCs and AI in IoT. BlockIoTIntelligence aims to achieve decentralized big data analysis considering security and centralization issues of IoT applications in various domains such as smart city, healthcare, and intelligent transportation. BlockIoTIntelligence claims the mitigation of existing challenges obtain high accuracy, reasonable latency, and security.

F. Approaches Similar to This Work

A use-case similar to this work [21] is the design and implementation of a camera-based sensor for room capacity monitoring. That work aims to count the number of people present in a room with the help of a Raspberry PI (RPI) [2] device equipped with a camera. Their architecture employs AI and IoT. The role of the camera is to take pictures. The pictures analyzed with a Machine Learning (ML). When the analysis completes, the data is sent to a LoraWAN Server [1]. To this end, they have attached a LoraWAN modem to the RPI. Furthermore, a web application is developed that shows the occupation of a given room. In that work, face detection is performed directly on the RPI. Finally, the algorithm counts the number of people entering the room and reports the number to the LoraWAN back-end server. That work still depends on a central server. Furthermore, the data is stored in a database. Finally, security issues are not considered.

G. The Newly Proposed Approach

This approach differs from related work implementations by providing a solid use-case in access management. Furthermore, this approach combines all three techniques at the same time, *i.e.,* AI for image processing, BC for immutable tamper-resistant storage (*e.g.,* for auditing reasons), and IoT for data harvesting (*i.e.,* providing the video stream). Furthermore, this approach follows the novel TinyML paradigm, in which face detection and recognition run directly on an IoT device.

III. Use-Case and Architecture

Driven by the specific use-case the description of the architecture follows as a generalized approach.

A. Use-case

The system employees real-time face detection and recognition of authorized individuals to grant access to an institution. The access is granted or denied by the system automatically. For example, when access is granted, the door to an institution may open automatically without any intervention. However, when access is denied, the door will remain closed, preventing the user from accessing a given resource. Every time an individual needs access, their picture is taken, processed, and stored in the immutable BC, preventing future tampering with data and enabling immutable storage that provides a solid foundation for auditing purposes.

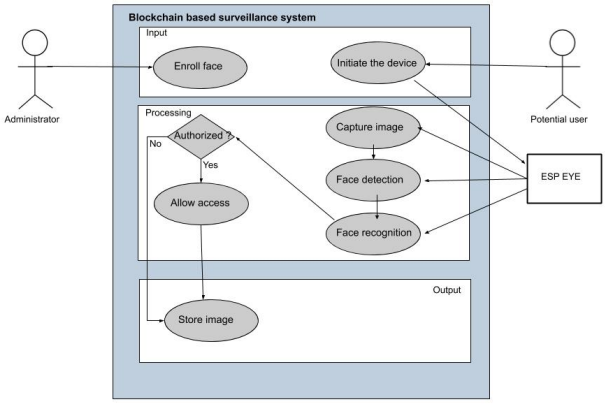


Fig. 1: High Level Use-case Overview

This use-case (*cf.* Figure 1) assumes that access is revoked, when an individual is not registered in the system. However, a picture of unknown individuals is still taken and stored in the BC for auditing purposes. Once a stranger is detected continuously, a warning is raised to inform the administrator about this issue. Simultaneously, the number of people entering and leaving an institution or a room can be tracked. Tracking the number of people entering or leaving the room can be very beneficial in many situations, such as capacity tracking, infection spread.

The process involves an individual approaching an entry point of an institution or a room, where an IoT device equipped with a camera detects and recognizes the person and grants or rejects access. Face recognition is performed with the help of face alignment, detection, and recognition. The images processed are forwarded to the BC, where they are stored in an immutable data structure. The system raises alarms, if a continuous presence in front of the device is detected, where a person may want to perform a security breach.

B. Architecture

Based on hardware components available, the software architecture was designed and reasons for these design decisions taken to materialize the idea of IoT-based AI surveillance with BC are provided. Furthermore, different approaches, technologies, and communication protocols, considered throughout the design decisions, are described.

1) Hardware Components: The image capturing and face recognition are handled by Esp Eye [11] directly, whereas the BC environment is based upon an Intel-based machine running a macOS. Figure 2, top left-hand side, displays multiple Esp Eye devices, which communicate with the IoT gateway within the same WiFi based (*i.e.,* IEEE 802.11) network. The

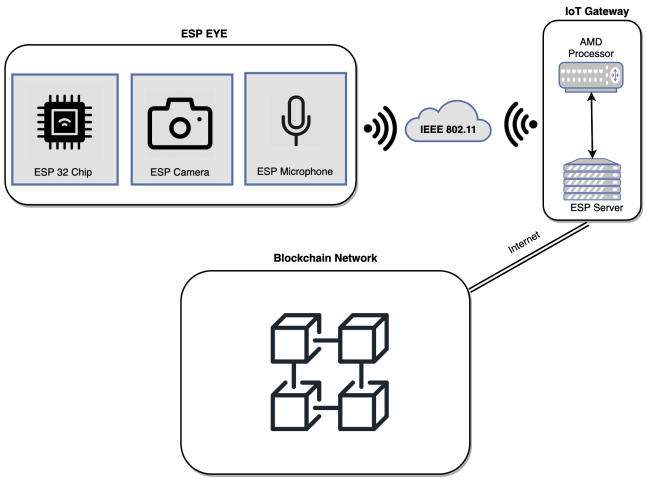


Fig. 2: Hardware Architecture

IoT Gateway serves as the middle man, which waits for data (*i.e.,* images and metadata) coming from Esp Eye devices to be inserted into the BC. The communication between Esp Eye devices and the IoT Gateway is achieved through a wireless communications compliant to the IEEE 802.11 Local Area Network (LAN) protocol. To provide integrated experimental facilities, the decision was to run a BC locally within the platform provided. However, the BC can be spanned among multiple machines organized as a BC network on the Internet without any hassle. Since HyperLedger Fabric (HLF) [5] was selected as the BC platform and its official build is provided for Intel-based CPUs, the decision was to run the IoT gateway on an Intel-based machine. It is, however, expected that HLF may run on low capacity devices, such as ARM-based RPI devices.

To this end, the HLF developer (*i.e.,* IBM) shall provide an appropriate compilation environment to support ARM-based devices as well.

Esp Eye [11] integrates an embedded microphone and a camera with 2-Megapixels. The camera is an OV2640 sensor with a maximum image size of 1600×1200 pixels. The board supports 2.4 GHz WiFi technology to connect to the Internet through a Local Area Network (LAN). A Micro USB port provides the power supply and also allows for debugging. In addition, it comes with a Universal Asynchronous Receiver-Transmitter (UART) port, which enables asynchronous serial communication to program the Esp Eye device. Esp Eye also supports several security features, such as flash encryption and secure boot. The flash encryption is intended to secure the content of the flash memory. In Esp Eye, flash encryption is performed using AES-256, and the key is stored in the eFuse (*i.e.,* special-purpose storage on the chip). eFuse keeps the values intact and cannot be changed by a software. Furthermore, since the data on the flash is encrypted, a physical readout will not be possible. Furthermore, secure boot can protect the device from uploading unsigned code. The device uses a typical digital signature method with the Rivest-Shamir-Adleman (RSA) cryptography. The public key is stored in the device itself, whereas the private key is kept secret and used upon each code upload. Apart from security, Esp Eye is an outstanding device due to its performance (*i.e.,* a double-core architecture supporting the 240 MHz CPU frequency) and a face detection/recognition platform known as Esp Who. As elaborated in Section II-B, Esp Who comes with algorithms for face detection (*i.e.,* MTMN) and recognition (*i.e.,* FRMN), which both run directly on the Esp Eye device.

2) Software Architecture: Figure 3 depicts a high-level overview of the software components. It is essential to mention that the software design shall be compatible with many underlying hardware architectures. However, the Esp Eye is required for the success of this project.

At the beginning, two approaches were considered: deployment of the face detection/recognition on the cloud and on the Esp Eye itself. In the first approach, the image processing AI is deployed on the cloud, which means the camera of a sensor captures images and sends them toward the cloud for face recognition. However, this centralized approach suffered from a single-point-of-failure and was later discontinued. In this work, Esp Eye uses the Esp camera to capture images. Images are sent to the face detection and face recognition modules. It is noting-worthy that face detection (*i.e.,* MTMN) and face recognition (*i.e.,* FRMN) run directly on the Esp Eye device. The outcome, *i.e.,* an images accompanied with metadata, is provided toward the video streaming service, which connects with the Esp Server running on the IoT Gateway. The Esp Server provides the image toward the ESP Plugin (*cf.* the IoT Gateway), with the help of the HLF SDK, to submit the Transaction (TX) to the immutable ledger. HLF stores images coming from Esp Eye devices in immutable storage. HLF runs typical services required to run a BC. It consists of identity management, access control, and â consensus mechanism.

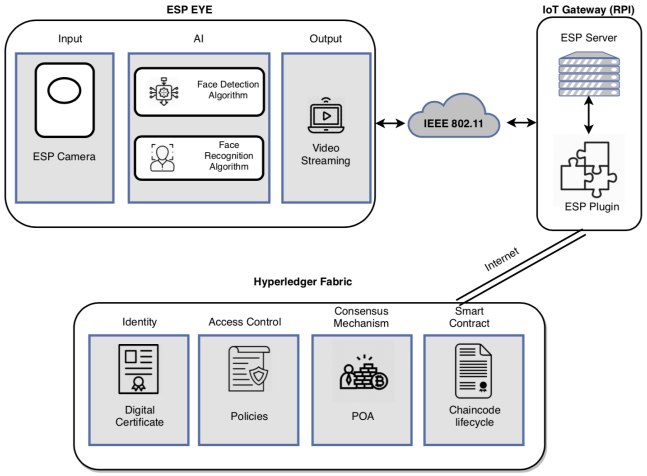


Fig. 3: Software Architecture

Finally, HLF maintains a smart contract, which is responsible for handling data received from its clients (*i.e.,* Esp Plugin residing on the IoT Gateway). HLF matches very well the use-case (*cf.* Section III-A), because a private permissioned BC is better suited for video surveillance due to privacy and BC block size reasons. The data gathered by a given organization have to remain protected against malicious third parties. This work also considered other BCs, for example, Ethereum (ETH) [30]. Although ETH is a public permissionless BC, it can still be adapted to the needs of this work (*e.g.,* by running the private network or encrypting data on the chain). However, ETH comes with a Proof-of-Work (PoW) consensus mechanism, which is energy demanding and costly. In contrast, HLF uses a modular architecture; it enables flexibility in selecting pluggable consensus mechanisms from a broader spectrum of candidate algorithms. Furthermore, HLF is fast, while some configurations allow for high workloads exceeding 20,000 supported TXs per second [13]. Additionally, HLF does not involve additional costs in terms of TX execution. Finally, Hyperledger Fabric allows for splitting the network into multiple smaller networks (*i.e.,* channels) to separate competing tenants and further improve privacy.

3) Communication Protocols: Considering communication protocols, Esp Eye is attached to a WiFi network, which provides Internet Protocol (IP) communication. This work considered two major protocols, *i.e.,* Message Queuing Telemetry Transport (MQTT) and Hypertext Transfer Protocol (HTTP), for the actual data transmission on the application layer, both supported by the Esp Eye device.

MQTT is a push-based client-server protocol that follows the Publish/Subscribe (Pub/Sub) communication model. Furthermore, MQTT is data-centric. Due to the fact that video surveillance works with images, the decision was to go towards document-oriented communication instead.

HTTP is a document-oriented client-server communication protocol. HTTP is employed between Esp Eye and the IoT Gateways to communicate in the client-server architecture. The

Representational State Transfer (REST) architectural style is used for inter-machine communication because REST is considered a lightweight communication paradigm. The JavaScript Object Notation (JSON) data interchange format [20] is employed to carry the actual information in the system. JSON is a lightweight format of syntax following the Javascript language notation. It makes JSON easy to read as well as process with high-level languages such as Javascript (JS).

C. Implementation

The implementation determines the data flow between Esp Eye, IoT Gateway, and HLF as shown in Figure 4, where data circulates from the left (*i.e.,* the Esp Eye device) to the right (*i.e.,* HLF) [**?**]. Several Application Programming Interfaces (API) and data structures are used to materialize the system.

1) Esp Eye Transmission Overview: Esp Eye analyses each frame with the MTMN face detection and if a face is detected, the image is provided towards the FRMN face recognition algorithm. Hence if a face is detected, but not necessarily recognized, the HTTP client is activated, while Esp Eye devices act as clients communicating with the remote server on the IoT Gateway. The video service takes the image equipped

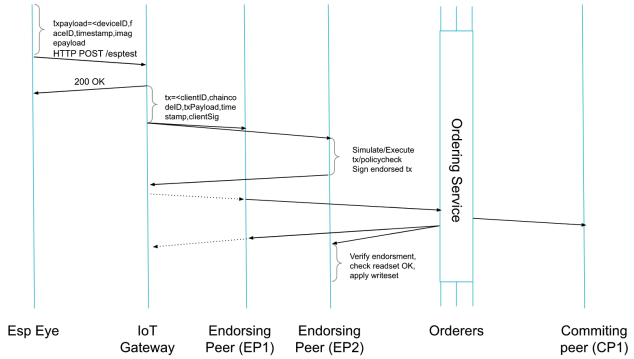
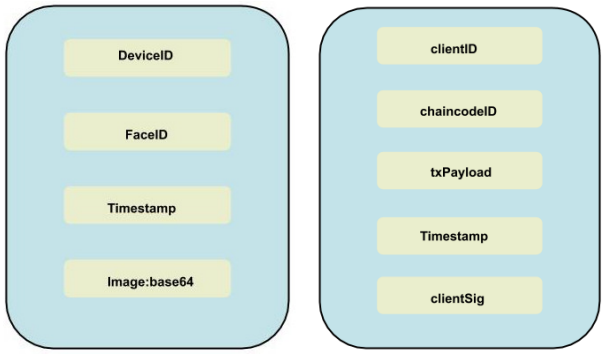


Fig. 4: Sequence Diagram Showing TX Execution

with meta-data (*i.e.,* device id, detected face id, timestamp) and forwards it to the IoT Gateway. This is achieved with the help of a REST API call using the HTTP POST request. The node.js [29]-based server located in the IoT Gateway receives the request (*e.g.,* an image with the details of a person detected provided as meta-data). Furthermore, the role of the node.js server is to properly acknowledge the successful reception of the transmission coming from Esp Eye devices.

There are four significant parameters to be stored in the BC reflected in the JSON document (*cf.* Figure 5a) provided by Esp Eye devices. First, as multiple Esp Eye devices may be employed in access management, the device id is essential, since the framework needs to distinguish particular devices from which the information is coming. Second, a face id is needed, allowing for personal identification without processing the captured frame again. The successfully identified person on the sensor implies that access was granted to given resources protected by this access management system. Additionally, the timestamp identifies the time moment when the person is detected. Finally, the image frame is encoded in BASE64 [15].

As shown in Figure 4 all four parameters are sent as a JSON document. After receiving the document, the node.js server located on the IoT Gateway responds with a status code.



(a) (b)

Fig. 5: JSON Formats Used: (a) Sent From Esp Eye Devices, (b) Included in the HLF TX Proposed

The Esp Eye is programmed with the help of the Arduino IDE [8], which has to be equipped with the ESP32 board support. Consequently, the appropriate board called ESP-CAM has to be selected to program the Esp Eye device. The WiFiClient library[1] enables establishing the connection to the IoT Gateway by using its IP address. The Esp Eye issues HTTP requests using the POST method towards the HTTP API exposed by the the IoT Gateway. For submitting JSON documents, the *application/json* method is used, which is a standard format for sending structured data. JSON is handful for sending plain text or any other data types. Since HLF also uses a JSON format to store assets, this work converts the image into a data type supported by JSON as well. Hence, the best option is to store the image as a string. To this end, the image is first converted to BASE64. The Esp Eye converts images to BASE64 and encapsulate the BASE64 representation into a JSON format.

2) HLF Transaction Submission: The node.js HTTP server is responsible for receiving the data and forwarding it to the HLF. In order to interact with HLF, the node.js-based server uses the HLF SDK providing an API to submit TX to the ledger. The process of submission takes place right after the image has arrived from an Esp Eye device. The HLF TX data, also referred to as an asset, is a collection of key-value pairs. Since the JSON format for the image transfer is used from the start, node.js may forward a similar JSON file to the BC.

Before the HLF TX is initiated three preconditions have to be fulfilled. *(i)* The HLF has to be configured and running. *(ii)* The HLF channel is established (*i.e.,* HLF sub-network for privacy concerns). *(iii)* The chaincode (*i.e.,* HLF Smart Contract identified by its chaincode id) with its endorsement

[1]https://www.arduino.cc/en/Reference/WiFiClient

policy is deployed within a given channel. Every Esp Eye is registered and enrolled with Certificate Authority (CA), while the Membership Service Providers (MSP) maintain identities of every Esp Eye device. For more information, please consult the HLF documentation [5].

The IoT Gateway acts as an HLF client. Therefore, the IoT Gateway initiates the HLF TX with these four parameters provided as a JSON document by Esp Eye (*cf.* Section III-C1), which becomes the TX payload. Each Esp Eye is assigned with a public/private key pair, which is used to sign the HLF TX. In this work, the IoT Gateway retrieves the key pair of a given Esp Eye device and signs the TX on behalf of the device using the HLF SDK. Although TX signing is possible in the offline mode, meaning locally on the Esp Eye device, this was not implemented here, since the HLF client SDK is not ported for Esp Eye devices yet.

Typical BCs, such as ETH, use the order-and-execute paradigm, meaning the TX is first ordered in blocks and later on executed by all peers in a given BC network. The speed of the HLF environment is achieved through the application of a so called execute-and-order TX submission scheme, which depends on the configurable number of endorsement peers, which first execute all TXs received. Endorsers endorse a given TX. When the HLF TX receives endorsements, which satisfies the configured endorsement policy, the TX can be ordered in blocks by the ordering service according to a configured consensus mechanism and committed into the ledger by committing peers.

This work configures two endorsers (*i.e.,* EP1 and EP2, *cf.* Figure 4). They receive the TX proposal, which includes a client id, the chaincode id (*i.e.,* HLF smart-contract id), the TX payload, a timestamp, and the clientSignature as presented in Figure 5b. Only endorsing peers specified by the chaincode receive this TX proposal. The endorsement policy in this work requires both endorsement peers to endorse the received HLF TX before the TX might be submitted toward the ordering service. First, EP1 and EP2 check the format of the TX. Second, every TX has to posses a valid signature of a client appropriately registered within the MSP. Third, the client has an authorized member of the HLF channel. When all conditions have been checked, the endorsers invoke the chaincode using the JSON document received. Eventually, the TX is executed, however, it does not yet update the ledger. Now, the endorsers sign the proposed TX and send it back to the HLF client on the IoT Gateway. The intent of the HLF client is to submit the TX to the ordering service and update the ledger. If the HLF client's intention was to query the ledger, there is no need to submit anything to the ordering service.

Before the HLF submits the final version of the TX, HLF clients send the TX endorsed toward the ordering service. Now, the TX is equipped with signatures of endorsing peers. While the ordering service may receive TXs from other clients or ESP devices, the ordering service orders TXs according to a sequence number and packages them into blocks. When the maximum number of TXs allowed in a block is reached or the maximum block-time has passed blocks are sent to committing peers to be included in the ledger for an immutable storage.

Upon receiving a broadcast message with the created block from the orderers, committing peers verify the signatures of ordering nodes within a given block. HLF allows for configuring committing peers. However, typically all peers in a given channel may update the ledger. If the committing peers fail to verify the signature of ordering peers, the ledger will not include and rejects the newly created block.

IV. Evaluation

The most important criteria for an evaluation are reliability of the overall design, end-to-end delay from the moment the image is taken until it is finally submitted into the BC, quality of images captured by Esp Eye as well as stored in Hyperledger Fabric, and the energy efficiency of the solution from an IoT device's perspective. The evaluation integrates two Esp Eye sensors (dual-core Tensilica LX6 processor with a maximum frequency of 240 MHz, 8 MB PSRAM, and 4 MB flash) and regular macOS based computer having 2-core Intel Core i5 running at 2.7 GHz, 512 GB SDD disk, and 8 GB RAM. To begin testing, several steps are recommended. *(i)* The Esp Eye sensors are powered up using an external charger power bank; 10 face profiles are uploaded on the device. *(ii)* HLF is started with the help of docker containers. Currently, one committing peer and two endorsing peers are supported. The ordering service is set to *solo*, *i.e.,* one node submitting HLF TXs. The channel is configured and JS-based chaincode is deployed. *(iii)* The IoT Gateway starts the node.js-based HTTP server listening on port 8585.

A. Image Quality

The OV2640 camera embedded in Esp Eye is also supported by the Esp Who platform. The camera can be configured in terms of frame size and pixel format. The frame size may be set to one of the following options: FRAMESIZE_CIF (400x296), FRAMESIZE_QVGA (320x240), FRAMESIZE_VGA (640x480), FRAMESIZE_SVGA (800x600), FRAMESIZE_XGA (1024x768), FRAMESIZE_SXGA (1280 x1024), FRAMESIZE_UXGA (1600x1200).

From the above mentioned frame sizes available for the OV2640 camera, not all are suitable for face recognition. One of the main reasons behind that is a 3 dimensional matrix allocated for all 3 channels. Each frame in the memory requires 3 channels, where every pixel takes 4 bytes of float data type. Thus, FRAMESIZE_UXGA requires 23.05 MB, which exceeds the PSRAM capacity of the sensor. Therefore, this makes it impossible to allocate memory just for the image without considering the fact that face detection algorithm itself generates multiple candidate frames for detection. Furthermore, FRAMESIZE_XGA, FRAMESIZE_SXGA, and FRAMESIZE_UXGA have been tested and memory allocation failed. Thus, those formats are not considered for face detection and recognition. Starting with FRAMESIZE_SVGA, the memory could be allocated with the image size of 5.76 MB. With FRAMESIZE_QVGA and FRAMESIZE_CIF, the Esp Eye with face detection and recognition runs smoothly

and both perform almost equally against quality measures. One difference between them is the number of frames per second delivered. The larger the image size, the more processing time it takes to receive the image. Therefore, due to higher width the FRAMESIZE_CIF consumes more processing power, and it can achieve on average 3.2 fps (*i.e.,* 312 ms to deliver an image) whereas with FRAMESIZE_QVGA, the sensor can deliver 5.2 fps (*i.e.,* 190 ms to deliver an image).

B. Processing Delay of Face Detection and Recognition

First of all, the idea of performing face detection and recognition on Esp Eye is a novel approach because typically face detection and recognition run either on the cloud or on a local computer. In related work (*cf.* Sect. II, it was reported that face detection and recognition offloading to the cloud takes at least 2 s and sometimes goes up to 5 s. Furthermore, much more processing power and energy is required.

```
22:56:26.273 -> MJPG: 174ms (5.7fps), AVG: 174ms (5.7fps), 121+52+0=173
22:56:26.273 -> Getting a frame in 0 ms.
22:56:26.454 -> MJPG: 173ms (5.8fps), AVG: 173ms (5.8fps), 120+51+0=172
22:56:26.454 -> Getting a frame in 0 ms.
22:56:26.632 -> MJPG: 173ms (5.8fps), AVG: 173ms (5.8fps), 120+51+0=171
22:56:26.632 -> Getting a frame in 0 ms.
22:56:26.810 -> MJPG: 175ms (5.7fps), AVG: 175ms (5.7fps), 122+51+0=174
22:56:26.810 -> Getting a frame in 0 ms.
22:56:26.955 -> MJPG: 176ms (5.7fps), AVG: 176ms (5.7fps), 123+51+0=175
22:56:26.955 -> Getting a frame in 0 ms.
22:56:27.164 -> MJPG: 177ms (5.6fps), AVG: 177ms (5.6fps), 124+51+0=176
22:56:27.164 -> Getting a frame in 0 ms.
22:56:27.455 -> FACE ID ALIGNED
22:56:28.124 -> There is a person recognized and their matched ID: 0
22:56:28.124 -> MJPG: 978ms (1.0fps), AVG: 978ms (1.0fps), 124+167+684=977
22:56:28.124 -> Sending image to Hyperledger Fabric
```

Fig. 6: A Log From the Serial Port Showing the Running Time on Esp Eye

Therefore, face detection and recognition takes on average 1 s on Esp Eye, but with less energy and processing power than the cloud offloading method, however, it leads to the same results. Figure 6 depicts a log of the processing time by calculating the total time in ms. For instance, in case of the first image, all processing needs 173 ms to accomplish, which results in 121 ms to receive the image from the sensor, 52 ms to perform the face detection with MTMN, and 0 ms to perform the face recognition with FRMN. Since no face has been detected by MTMN, therefore, FRMN was not activated, which results in 0 ms completion time. Within the same log, at the end, a situation is displayed when a face was in fact detected and recognized. In this case the third value is not 0, but it holds a significant value of 684 ms. Therefore, the total time including face detection (*i.e.,* MTMN) and recognition (*i.e.,* FRMN) results in 977 ms, which includes the image acquisition, MTMN, and FRMN. The FRMN algorithm displays around 99% percent accuracy, however, more studies are needed to evaluate its performance on face detection in a real-system.

C. End to End Processing Delay

Many BC platforms suffer from a low TX rate. However, the HLF executes TXs in the execute-and-order way. Thus, HLF gains significant speed in terms of processing delay and supports large TX volumes per second. This work measures the end-to-end delay by printing the timestamp at different individual processing steps. *(i)* The image is captured, but no face detection/recognition has been performed yet. *(ii)* A face has been detected/recognized and the image is sent to IoT Gateway. *(iii)* The image has reached the IoT Gateway. *(iv)* The image has been submitted to HLF. *(v)* The image is inserted in the ledger and the ordering service is finished.

Table I shows all processing stages starting with the Esp Eye image capturing until the image reaches the ledger. Similarly, as it was estimated earlier, face detection and recognition takes on average 1 s, however, if the face detection phase is not executed, the image processing is much shorter (*i.e.,* at around 200 ms). Sending the image from Esp Eye to the IoT Gateway takes almost 2 s. Furthermore, the time it takes from the IoT Gateway until the TX is submitted to HLF is very small in comparison to other BCs. HLF consumes little more than 2 s. This is influenced by two configurable parameters, *i.e.,* BatchTimeout and BatchSize. BatchTimeout is important for this work as it refers to a block-time limit. This work configures BatchTimout at 2 s, which minimizes the idle time of ordering peers. However, depending on the use-case, those parameters may need to be tuned such that there is no bottleneck in TX arrivals. The end-to-end delay experienced in the system is 5.3 s from the moment image is taken until the inclusion of the BC TX in the BC. Therefore, almost real-time use-cases can be supported using HLF as a communication backend.

TABLE I: End-to-End Delay Measurements

Action Point	**Timestamp**
Image is captured by Esp Eye	**2021-01-22T14:39:26Z**
Image is being sent to IoT Gateway	**2021-01-22T14:39:27Z**
Image is received by IoT Gateway	**2021-01-22T14:39:28.947Z**
Image is submitted to Fabric	**2021-01-22T14:39:29.111Z**
Image has reached all the peers	**2021-01-22T14:39:31.313Z**
Total Time	**5.313 s**

D. Energy Efficiency of Esp Eye

The experimentation setup was used to measure the energy consumption of Esp Eye against image quality, face detection, and face recognition. After several tests with different image qualities and parameters, there was no difference experienced in Esp Eye energy consumption. Throughout the experiment, the energy consumption remained at the constant level of 600 mW (in total, the device consumed 600 mWh within an hour of operation), which allows for a 7-hour operation on an alkaline battery of 4,200 mWh capacity. Furthermore, there is no difference in energy consumption when a face is recognized or no face is detected.

V. Summary, Discussion, and Future Work

This paper provides the first access management system, which utilizes Artificial Intelligence (AI), Blockchains (BC), and Internet-of-Things (IoT) in an integrated use-case. This use-case is centered around access management without direct

human intervention. Thus, the user needs to present his/her face in front of a camera to access a resource. The system takes the image of that person and checks, whether this given user has the right to access a given resource. Face detection and recognition are performed directly on the IoT device. To detect faces, a MT-CNN (Multi-Task Cascaded Convolutional Network) with MobileNets (MN) was deployed. Furthermore, the Face Recognition model is based upon a Convolutional Neural Network (FRMN). To establish a good level of transparency, the AI decisions on access rights as well as images taken by the sensor are stored in the immutable, tamper-resistant storage implemented with the help of the HyperLedger Fabric (HLF). The performance of the system was evaluated at an excellent level, where a 5.3 s end-to-end delay is reached. This value reads to be outstanding compared to well-established BC systems, such as Bitcoin (BTC) or Ethereum (ETH) having a block time configured at the level of 10 min and 10 s, respectively. Additionally, the size of data to be persisted inside a block does not lead to high costs, as for a BTC or ETH case, since HLF does offer "unlimited" storage capacity due to its private, permissioned characteristic. At the same time, data privacy is ensured, since the private ledger within the HLF implementation will only allow for authorized accesses by definition.

As this approach taken is considered to be a prototype, two current drawbacks will have to taken care of in the next generation:

1) All face descriptors of recognized faces are hard-encoded on Esp Eye devices.
2) The HLF Transaction (TX) signing is not performed directly on Esp Eye devices, since it is delegated to the IoT Gateway device supporting this process on behalf of the sensor.

Besides solving these aspects, future works will not only close the gaps in full as identified in the currently specified and prototyped system, but will further develop the IoT-BC integration into operational and reusable software functionality:

1) The face descriptor management will allow an IoT device to retrieve a remote directory of recognized faces.
2) Porting the HLF Software Development Kit (SDK) to IoT devices will enable for the HLF TX submission directly from IoT devices.

ACKNOWLEDGEMENTS

This paper was supported partially by *(a)* the University of Zürich UZH, Switzerland, and *(b)* the European Union's Horizon 2020 Research and Innovation Program under Grant Agreement No. 830927, the CONCORDIA project.

REFERENCES

[1] "A Technical Overview of LoRa and LoRaWAN," https://lora-alliance.org/portals/0/documents/whitepapers/LoRaWAN101.pdf, last visit: May 17, 2019.

[2] "Raspberry Pi 3 Model B." https://www.raspberrypi.org/products/raspberry-pi-3-model-b, last visit: October 2, 2019.

[3] S. F. Abedin, M. G. R. Alam, R. Haw, and C. S. Hong, "A system model for energy efficient green-iot network," in *2015 International Conference on Information Networking (ICOIN)*, 2015, pp. 177–182.

[4] D. Ackerman, " System brings deep learning to "internet of things" devices," https://news.mit.edu/2020/iot-deep-learning-1113, [Online; accessed 10-December-2020].

[5] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3190508.3190538

[6] H. F. Atlam, R. J. Walters, and G. B. Wills, "Intelligence of things: Opportunities challenges," in *2018 3rd Cloudification of the Internet of Things (CIoT)*, 2018, pp. 1–6.

[7] H. F. Atlam, M. A. Azad, A. G. Alzahrani, and G. Wills, "A review of blockchain in internet of things and ai," *Big Data and Cognitive Computing*, vol. 4, no. 4, 2020. [Online]. Available: https://www.mdpi.com/2504-2289/4/4/28

[8] Y. A. Badamasi, "The working principle of an arduino," in *2014 11th International Conference on Electronics, Computer and Computation (ICECCO)*, 2014, pp. 1–4.

[9] A. Banafa, "IoT and Blockchain Convergence: Benefits and Challenges," *IEEE Internet of Things*, Jan. 2017.

[10] T. N. Dinh and M. T. Thai, "Ai and blockchain: A disruptive integration," *Computer*, vol. 51, no. 9, pp. 48–53, 2018.

[11] Espressif, "ESP-EYE Development Board," https://www.espressif.com/en/products/devkits/esp-eye/overview, [Online; accessed 10-December-2020].

[12] S. Gopinath, N. Ghanathe, V. Seshadri, and R. Sharma, "Compiling KB-Sized Machine Learning Models to Tiny IoT Devices," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 79–95. [Online]. Available: https://doi.org/10.1145/3314221.3314597

[13] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, "Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019, pp. 455–463.

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[15] S. Josefsson *et al.*, "The BASE16, BASE32, and BASE64 data Encodings," IETF RFC 4648, October, Tech. Rep., 2006.

[16] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *IEEE Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[17] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," *arXiv preprint arXiv:2007.10319*, 2020.

[18] S. R. Niya, E. Schiller, I. Cepilov, and B. Stiller, "BIIT: Standardization of Blockchain-based I2oT Systems in the I4 Era," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–9.

[19] B. Parker and C. Bach, "The synthesis of blockchain, artificial intelligence and internet of things," *European Journal of Engineering and Technology Research*, vol. 5, no. 5, pp. 588–593, May 2020. [Online]. Available: https://www.ejers.org/index.php/ejers/article/view/1912

[20] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of JSON Schema," ser. WWW '16. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2016, p. 263–273. [Online]. Available: https://doi.org/10.1145/2872427.2883029

[21] S. A. I. Quadri and P. Sathish, "Iot based home automation and surveillance system," in *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, 2017, pp. 861–866.

[22] E. Schiller, E. Esati, S. R. Niya, and B. Stiller, "Blockchain on msp430 with ieee 802.15.4," in *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, 2020, pp. 345–348.

[23] E. Schiller, S. R. Niya, T. Surbeck, and B. Stiller, "Scalable Transport Mechanisms for Blockchain IoT Applications," in *2019 IEEE 44th LCN Symposium on Emerging Topics in Networking (LCN Symposium)*, 2019, pp. 34–41.

[24] S. Shang, H. Liu, Q. Qu, G. Li, and J. Cao, "Frmn-a face recognition model based on convolutional neural network," in *IOP Conference Series: Materials Science and Engineering*, vol. 585, no. 1. IOP Publishing, 2019, p. 012101.

[25] S. Singh, P. K. Sharma, B. Yoon, M. Shojafar, G. H. Cho, and I.-H. Ra, "Convergence of blockchain and artificial intelligence in iot network for the sustainable smart city," *Sustainable Cities and Society*, vol. 63, p. 102364, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2210670720305850

[26] S. K. Singh, S. Rathore, and J. Park, "Blockiotintelligence: A blockchain-enabled intelligent iot architecture with artificial intelligence," *Future Generation Computer Systems*, vol. 110, 09 2019.

[27] O. Source, "ESP-WHO Overview," https://github.com/espressif/esp-who, [Online; accessed 23-September-2020].

[28] J. Tang, D. Sun, S. Liu, and J. Gaudiot, "Enabling deep learning on iot devices," *Computer*, vol. 50, no. 10, pp. 92–96, 2017.

[29] S. Tilkov and S. Vinoski, "Node. js: Using javascript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, 2010.

[30] M. Valenta and P. Sandner, "Comparison of ethereum, hyperledger fabric and corda," 2017.

[31] N. Waheed, X. He, M. Ikram, M. Usman, S. S. Hashmi, and M. Usman, "Security and privacy in iot using machine learning and blockchain: Threats and countermeasures," *ACM Comput. Surv.*, vol. 53, no. 6, Dec. 2020. [Online]. Available: https://doi.org/10.1145/3417987

[32] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.