CO3320 FINAL PROJECT REPORT

A New Privacy-Focused Architecture Proposal: Decoupling Service from Infrastructure of a Cloud-Enabled Home Security Camera

by **Zoltan Toth-Czifra**

Computing and Information Systems
2020 June
Self-study
Supervisor: None
Student registration number: **170214074**

Submitted as part of the requirements for the award of the Degree in Computing and Information Systems of the University of London.

Contents

1 Summary

Popular online services such as Facebook [1], Instagram [2], Google Search [3] handle an ever increasing size of personal data of their users. Users have little control and visibility of the personal data stored about them by these services. This poses privacy risks and locks users into the services by holding their data hostage.

To mitigate these issues, in this paper I'm proposing a new type of architecture that breaks the assumption that the service provider needs to own and operate the entire infrastructure of the service. The proposal gives the choice and full control of the data storage infrastructure to the user. To demonstrate the concept I implemented a prototype home security camera built around the concept of choice of the underlying cloud infrastructure.

2 Introduction

2.1 Background

The 2010s have brought a radical change to how people around the world think about big tech companies. While the change has been gradual, we can identify several pivotal events. In 2013 the global public was shocked by the revelations of Edward Snowden detailing how the NSA is cooperating with telecommunication companies and online services in order to implement mass surveillance. [4] It was made evident that popular online services such as Gmail or Facebook had been secretly spying on their users. In 2018 it became widely publicized that data analytics firm Cambridge Analytica harvested personal data from 50 million Facebook profiles, possibly with the deliberate collaboration of Facebook itself, and used that information to influence the U. S. presidential elections. [5]

An apt analogy comes to mind from the transportation industry. Governments around the world were generally not involved in car designs until the 1960s when the number of fatal car accidents started to increase sharply [6]. This is analogous to the state of the tech industry today, which still enjoys relative freedom of regulation, but the negative consequences of this freedom are already visible and lawmakers and engineers are starting to pay attention.

2.2 A new architecture

In this paper, I am proposing a new kind of service architecture that breaks previous assumptions about how online services can operate, and I will demonstrate it through a home security camera device and mobile application developed for this project. The proposal goes beyond this particular application and has the potential to improve online privacy of other services, and increase control over the user's own data and provide data portability generally. While the proposal is technical in nature, it has the capacity to form part of a legislative solution. In that, it is akin to the most effective combination that drastically reduced fatal car accidents made up by a mix of regulation and engineering; the three-point seatbelt and the law that mandates its use.

From social networks to cloud email solutions, from smart home devices to cloud file storage services, nearly all online services operate under the following assumption: **the service and the underlying infrastructure are indivisible**. When a user signs up to a social network such as Facebook, starts using a cloud email service such as Outlook.com [7], or perhaps, as most relevant to this project, purchases a cloud-enabled home security camera from Google (NestCam) [8], they implicitly accept that by using the service itself, they commit to the underlying infrastructure operated by the same entity that runs the service or sells the hardware itself. In other words, when you upload your photos to Instagram, for example, you commit to

using Instagram's servers to store and serve them as shown on Figure 1.

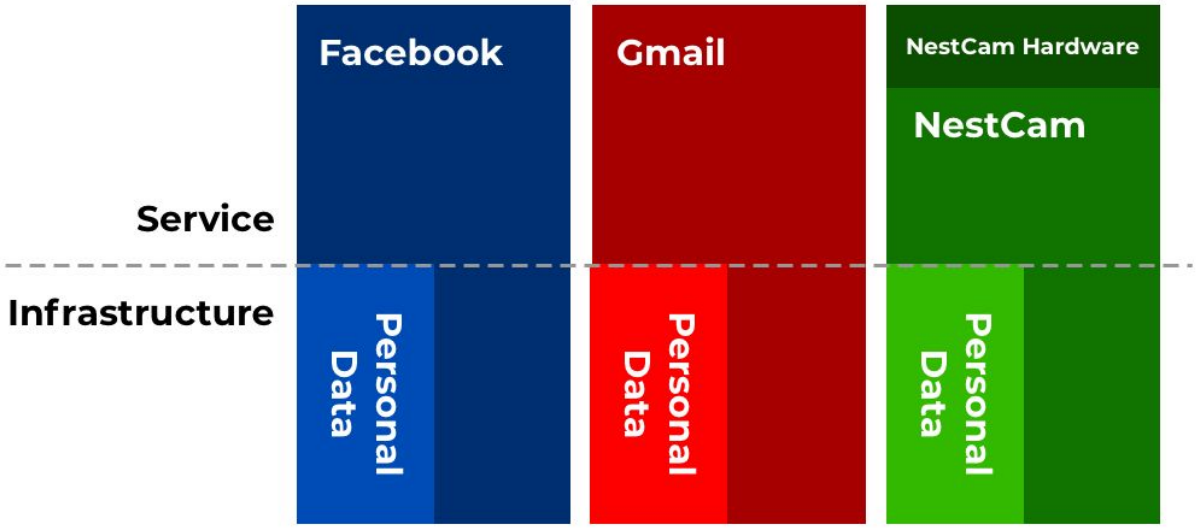


Figure 1: Diagram showing the strong coupling of the service and the underlying infrastructure of some online services.

At first, this assumption seems obvious and banal. How can the service exist without the underlying infrastructure? How can the infrastructure be operated by someone other than the operator of the service itself? How can the user of the service avoid locking themselves into the infrastructure by using the service?

We used to know the answers for these questions; some of the most successful technologies of the internet such as WWW, SMTP, IRC, etc (not an IT acronym) were built on the ideas of platform independence, open standards and most importantly, *choice*.

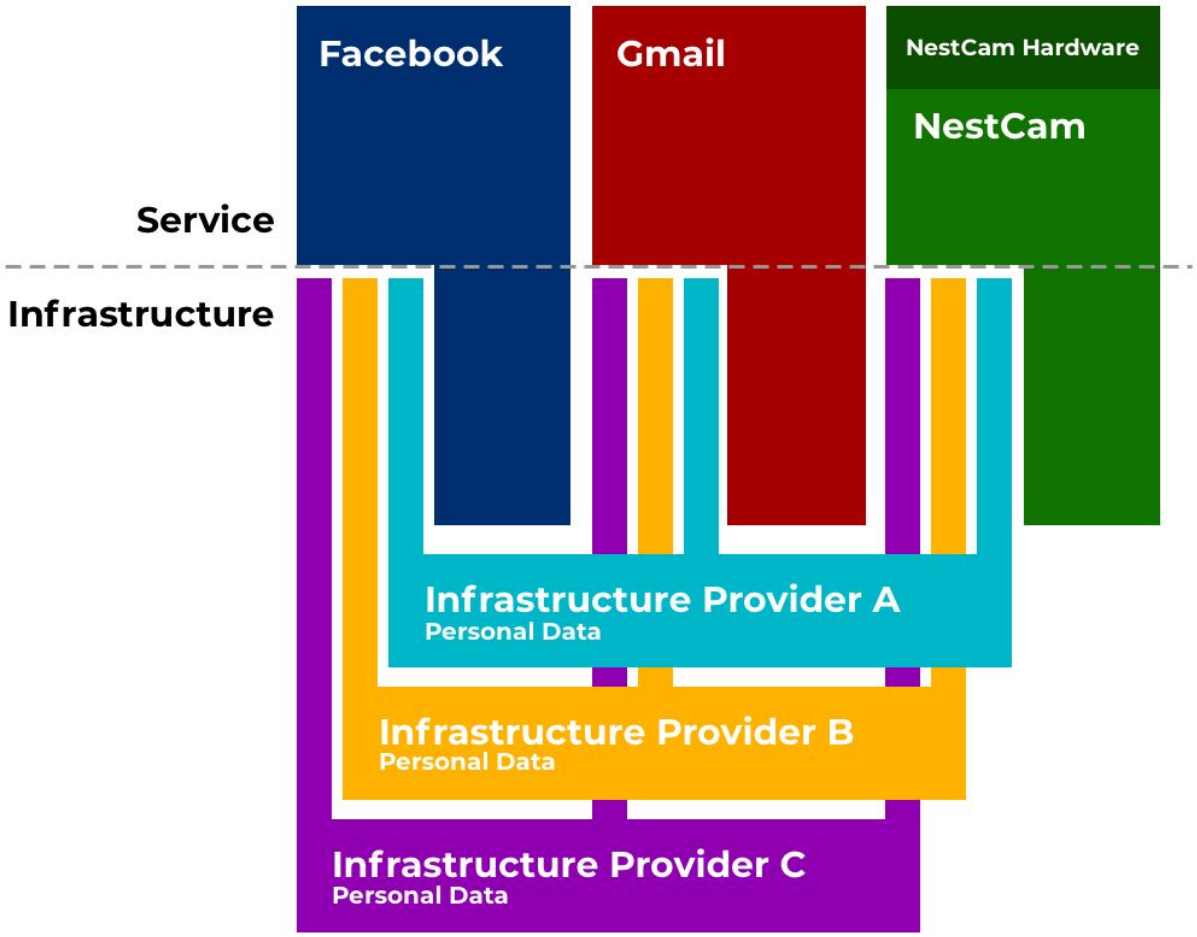


Figure 2: Diagram showing the proposed solution, abstracting away part of the infrastructure that concerns storing personal data.

In this paper I argue that there is nothing inherent in the privately owned, centralized online services of 2020 that make it impossible to separate their infrastructure from the service (Figure 2). I will attempt to demonstrate this with a real-life example.

2.3 The Demonstration Project

While the concept of separating infrastructure from service is widely applicable to a range of services, as a proof of concept, I have chosen to implement a relatively simple to do project: a home security camera with cloud syncing.

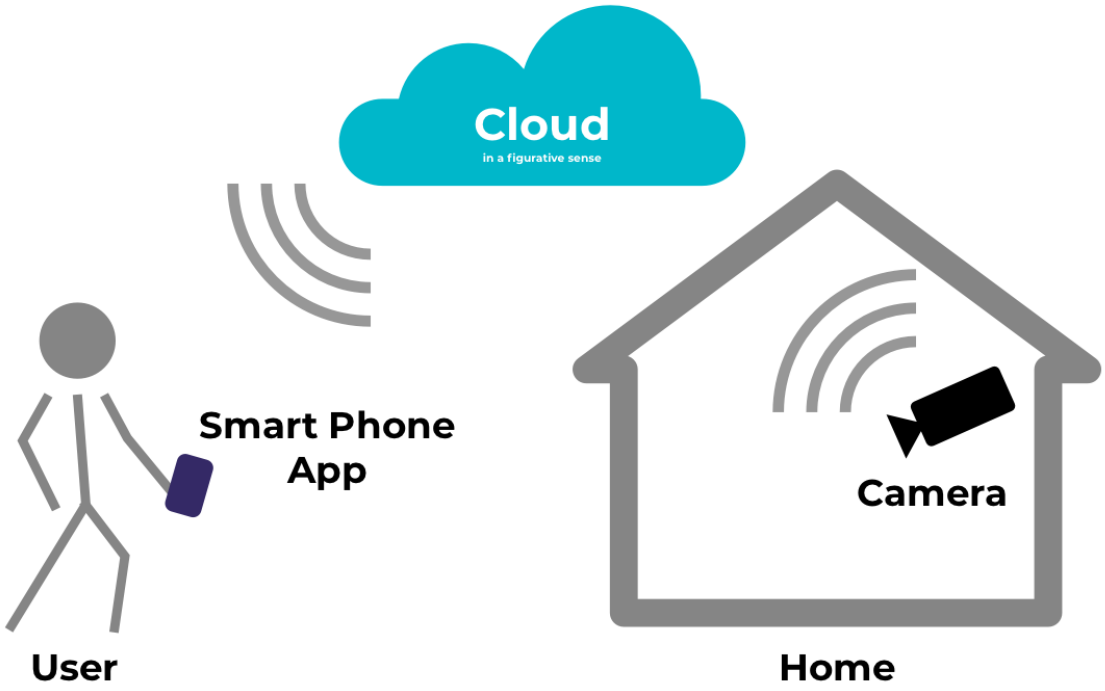


Figure 3: High-level view of the function of cloud-connected home security cameras

Internet-connected home security cameras (IP cameras) have become increasingly popular in the past years due to increasingly affordable hardware, ubiquitous wireless internet connections and the prevalence of smartphones to control and monitor these devices. These devices are typically always on, connected to the home computer network and synchronize a video and audio feed from the come to the cloud. The video and audio feed are accessible to the user from their smartphone either as a live playback (e.g. then they are away from home) or playback of a past recording (see Figure 3).

One of the larger players in this market is Nest and their Nest Cam [8] product. I chose this product to study because Nest has recently been acquired by Google [9], which makes it very relevant in terms of privacy concerns. This ownership relation is concerning due to Google's spotty privacy record. The fundamental issue is that Google, in order to provide the remote monitoring functionality that's key for a home security camera, stores and has access to the video feed recorded by the device. Arguably there are very few types of data more private than a direct video feed into someone's living room, yet many of us trust Google with full, unrestricted access to this. This includes the author - I happen to own one of these little spy machines.

The ultimate goal of this project is to construct a prototype that has similar functionalities to Nest Cam, but it's built using an architecture that lets the user choose (part of) the underlying cloud infrastructure. Very simply put: when setting up the camera, the user can choose where the security footage will be stored. There are several existing infrastructure providers listed (e.g. Amazon Web Services, Google Cloud, Digital Ocean) but the product allows implementing infrastructure connectors. Using this architecture, the provider of the service (the hardware and the software) doesn't need to have access to the data at all and the infrastructure where it's stored will be completely abstracted from them (see Figure 4).

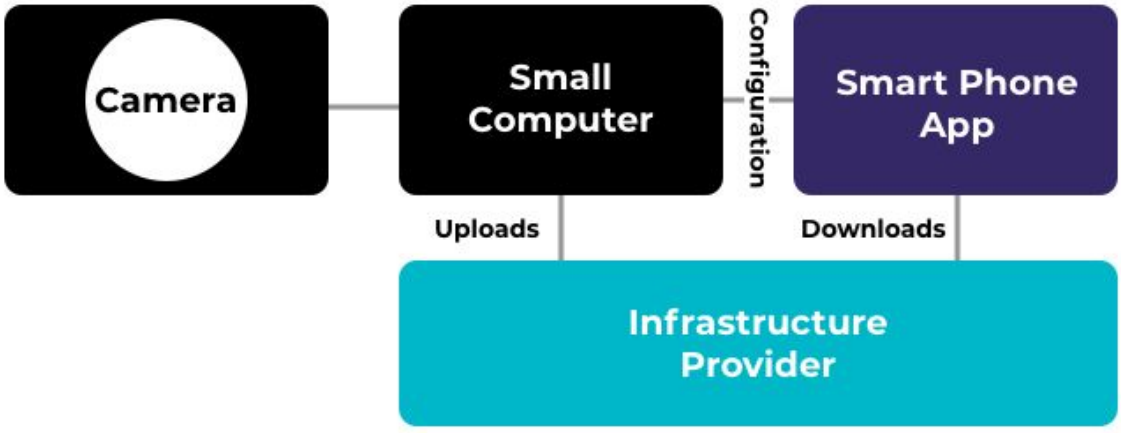


Figure 4: High-level overview of the project

Through this simple but practical example I am going to demonstrate that separating the service from the infrastructure is possible and it can have comparably positive effects.

3 Literature Review

In my literature research I was mostly focused on finding similar technical proposals that promise to mitigate privacy issues associated with online services, especially alternative architectures that fundamentally affect the service. While I haven't found literature on the exact same approach I'm proposing, I have found different approaches to solve the same problem.

3.1 Dweb

A collection of these proposals has been referred to by the umbrella term Decentralized Web (or sometimes Distributed Web) or Dweb. What is Dweb?

A 2018 article published in The Guardian titled *Decentralisation: the next big step for the world wide web [10]* sums it up well: "The DWeb is about re-decentralising things – so we aren’t reliant on these intermediaries to connect us. Instead users keep control of their data and connect and interact and exchange messages directly with others in their network."

The motivation behind Dweb projects is the same recognition that motivates my project. In a paper titled *A taxonomy of decentralized online social networks [11]* the authors write: "the servers are controlled by a single authority, thereby providing a single system image to the end users. This poses serious threats to the user privacy and content ownership."

The paper outlines two different competing concepts to implement Dweb services:

1. Peer-to-peer networks of end user devices sharing data directly with one another
2. A federation of multiple independent services that users can choose from

Most if not all Dweb services I covered in my research were a kind of online social network (OSN), similar to existing OSNs such as Facebook or Twitter. They are called Distributed OSNs or DOSNs.

3.1.1 Peer-to-peer services

In this architecture the users themselves directly share content with each other, typically from devices they own (e.g. smartphones, home computers). Some of the concrete projects, most notably PeerSon [12] and Safebook [13] propose a multi-tier architecture, where some of the metadata is stored in a separate tier from the actual data and provides lookup services, record keeping for offline data, etc. However, this upper tier is still implemented on top of a distributed

architecture between the peers, e.g. as a Distributed Hash Table (DHT), as opposed to a centralized service, although some implementations allow for a centralized service. In some DOSNs, such as SuperNova [14], while the personal data is still stored on devices owned by the users, caching of this data is done by "super peers" on the network to guarantee offline access.

The identity management of the peer-to-peer services is typically implemented through the public key infrastructure, although there are exceptions, e.g. PrPl [15] that relies on OpenID.

In my research I looked at several peer-to-peer OSNs . They all exhibit similar architectures: full decentralization without central control, optionally with some high-availability solution for the entire data set or just the metadata, as shown on Figure 5.

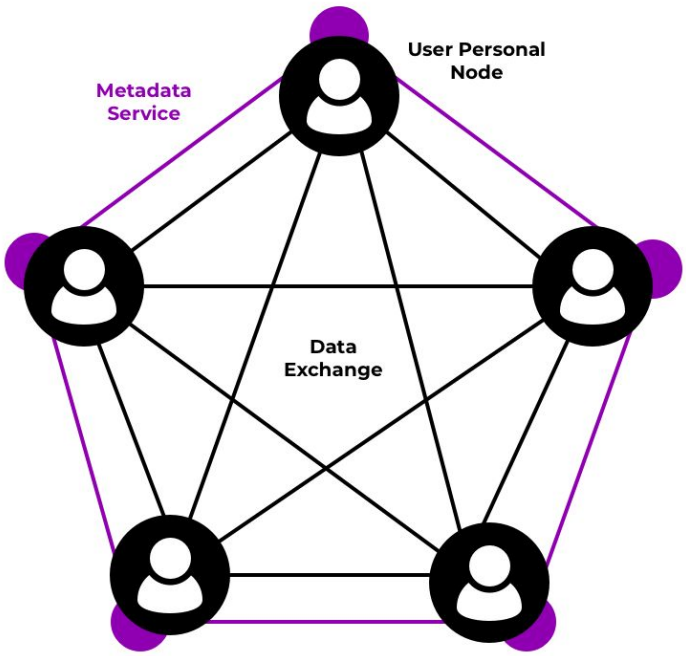


Figure 5: Peer-to-peer architecture with an optional metadata service tier

3.1.2 Federated services

Federated services differ from peer-to-peer networks in that there's a significantly smaller number of nodes than users on the network, and a node hosts many user accounts. The user selects a server when signing up to the service and only communicates with that server in the future, but all servers communicate with each other in a decentralized federation. Here the word *server* is to be understood conceptually; physically it might consist of a cluster of servers. This type of architecture is similar to some older successful protocols, like SMTP (emailing) or XMPP (instant messaging).

Some concrete projects, such as Vis-a-Vis [16] promotes hosting the data on infrastructure-as-a-service providers, such as Amazon Web Services. In that, it's somewhat closer to my architecture proposal than the pure peer-to-peer networks mentioned previously, but it's still distributed in nature and lacks any sort of centralized control. Anyone can add servers to the network. Perhaps the most successful DOSN, Diaspora [17], offers the same architecture, where anyone can host a "pod" and user data is propagated across the network of pods. In my literature review I examined a number of federated DOSNs, including microblogging services [18,19] and instant messaging services [20], all with similar architectures, shown on Figure 6.

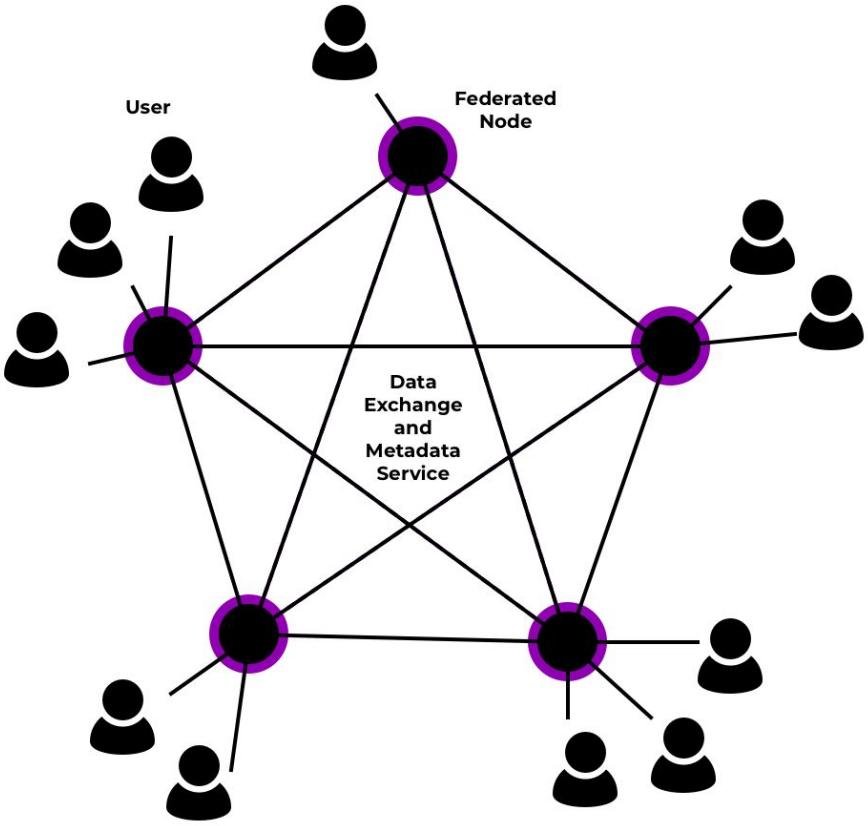


Figure 6: Federated architecture

3.1.3 Problems

"Decentralization is undoubtedly one of the inherent solutions to major offline privacy issues in OSNs; however, it does not come free of new challenges and issues to privacy itself." - summarizes the author of *Challenges in the Decentralised Web: The Mastodon Case [21]*. The study lists a number of privacy and fraud related issues with one popular Dweb microblogging service, Mastodon [19]. In a 2017 paper titled *Decentralized privacy preserving services for Online Social Networks [22]* the authors describe a number of problems with DOSNs:

- Performance is poor and availability is limited
- Access control becomes complicated
- Data replication poses privacy concerns
- Collaboratively owned content is difficult to implement

- Identity management, content moderation and fraud prevention are difficult to impossible

Another issue is the lack of a business model. "All of the DOSN proposals except SuperNova adopt the voluntary business model. In the voluntary model, a user either stores content in the leased node [...] or participates using his/her personal machine [...]. There is no monetary incentive involved in participation" [11]. This is a problem because some crucial OSN features do in fact require a functioning business model. Take content moderation, for example: it's a large cost element of traditional social networks and it cannot be automated or distributed. Additionally, Dweb solutions without a business model are limited in popularity to the number of industry enthusiasts, because they cannot market their products effectively.

Generally speaking control, a functioning business model and having (parts of) the service centralized (or better put: "un-distributed") are not necessarily evil things and could benefit the users of the service.

Yet another problem I found with Dweb projects I reviewed is that their focus is perhaps a bit too narrow on social networks, and the proposed architectures are not applicable in some domains. For example, my prototype product (security camera) cannot take advantage of a distributed architecture. Further, mandating decoupling of the infrastructure from the service (Figure 7) is a concept enforceable by regulation and can potentially work for many existing or new for-profit organizations and products, unlike Dweb solutions.

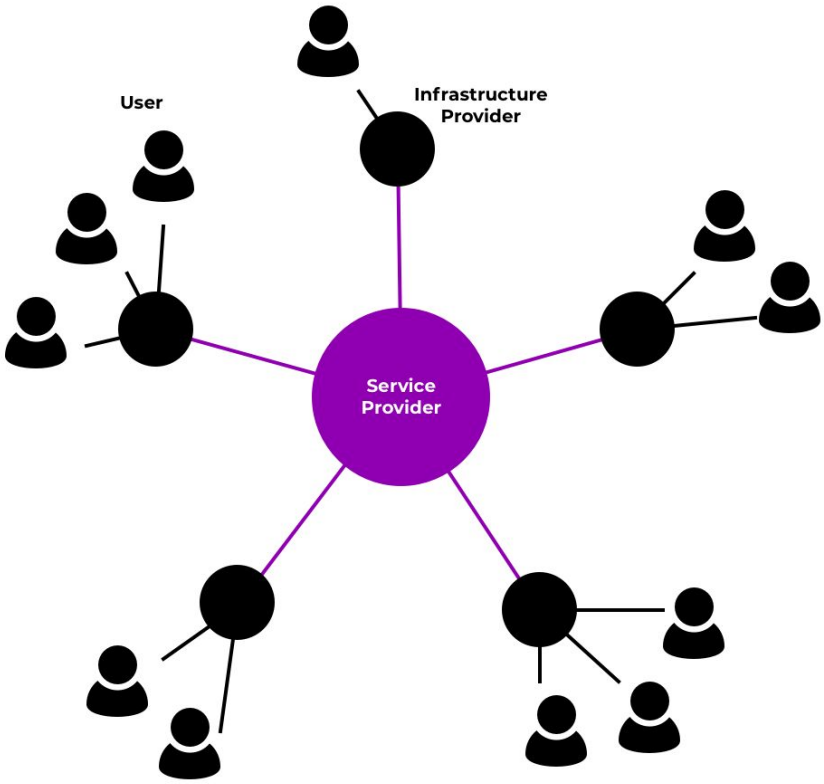


Figure 7: Separating infrastructure from service while keeping central control.

3.2 Other literature

While not academic in nature, a book that influenced my thinking about the subject is *Who Owns The Future?* [23] by legendary computer scientist and philosopher, Jaron Lanier. In his book Lanier goes into detail about the effects of technology giants on the middle class, the winner-take-all nature of so-called "siren servers" collecting ever more data of human subjects. He proposes an alternative structure different from those under Dweb, that returns control of data to the data subjects themselves (people who produced the data or who the data is about) and compensates them monetarily in proportion to the economic value their data creates. This proposal requires both technical and legislative paradigm shifts. While he admittedly sees little chance of his idea to be implemented in real life, his thinking about the subject largely inspired my project.

4 Methods

4.1 Who cares?

That question crossed my mind when I devised this project so I decided to verify the idea with an online survey. I put together a simple questionnaire of 17 questions using Google Forms. To obtain responses I ran a paid traffic campaign on Facebook, targeting users from the United States, United Kingdom and Spain (see Figure 8). I chose these countries because of their relevance to me or this project:

- Almost without exception the largest global tech companies are headquartered in the US
- The University of London is of course headquartered in the UK
- My permanent residence is in Spain

I do realize that obtaining traffic from Facebook adds a bias to the responses, since people who are most concerned about privacy probably won't use Facebook. Further, Facebook optimizes campaigns to goals such as traffic, conversion, etc. so the sampling is not random. Despite these imperfections, running a paid traffic campaign on Facebook seemed to be the best option to get a significant amount of responses from different targeted territories. Considering a total population size of approximately 439 million from the 3 targeted countries, I chose a statistical confidence level of 90% with 10% margin of error. This gave me a sample size of 69 responses, which seemed reasonably cheap to obtain through paid traffic. To calculate the sample size I used an online calculator [24].

Figure 8: Screenshot of the Facebook ad that was used to obtain survey responses (author not pictured, the photo is a public domain image from Pixabay.com)

In the end, I received 70 responses over the course of 4 days from 3 targeted countries (US, UK, Spain). I ran the survey in December 2019, before presenting my Preliminary Project Report, so that I could verify that I was working on a real problem before commiting to the project topic.

I will present the answers in the *5 Results* section, but I leave here the list of questions.

1. How concerned are you about online privacy in relation to large tech companies (Google, Facebook, Amazon, Apple, etc)? Scale 1-5; 1: Not at all, 5: Very concerned
2. How much do you trust large tech companies (Google, Facebook, Amazon, Apple, etc) to handle your private data responsibly? Scale 1-5; 1: Not at all, 5: Fully trust them
3. How much do you trust large tech companies to completely erase all your personal data when you remove your account? Scale 1-5; 1: Not at all, 5: Fully trust them
4. Have you ever deleted or considered deleting any of your social media or cloud storage (email, files) accounts due to privacy concerns? Options: Did not consider; Did consider but did not do it (yet); Did delete
5. What steps do you think need to be taken to improve online privacy in relation to large tech companies? Multiple choice options: More or better regulations by the government; Technical solutions that improve privacy; Tech companies should self-regulate; Users should stop using products from large tech companies; Other
6. Which of these services are you using actively? Multiple choice options: Google Search; Facebook; Twitter; Instagram; Pinterest; Reddit; Snapchat; Gmail; Hotmail or Outlook.com; Amazon (for shopping); Nest Cam
7. Do you own any home security device (e.g. camera) that uploads data to the cloud (e.g. a video feed)? Options: Yes; No; I don't know
8. Do you use any home assistant device like Google Home or Amazon Echo? Options: Yes; No; I don't know
9. How complete idea you have about what data is collected and kept about you by the large tech companies? Scale 1-5; 1: Very incomplete, 5: Fully complete
10. How complete idea you have about where and how the data collected about you stored by the large tech companies? Scale 1-5; 1: Very incomplete, 5: Fully complete
11. Would you be interested in seeing the complete raw data stored about you by large tech companies? Options: Yes; No; I don't know
12. Would you feel more comfortable using an online service if you could select where (in which country) your data is stored? Options: Yes; No; I don't know
13. Would you feel more comfortable using an online service if you could fully see your stored data at all times? Options: Yes; No; I don't know
14. Would you feel more comfortable using an online service if the company providing the service (e.g. Gmail, Facebook) was different from the one storing your personal data involved in using the service (e.g. your emails or photos)? Options: Yes; No; I don't know
15. Would you feel more comfortable using an online service if the company providing the service (e.g. Gmail, Facebook) if you could choose from a list of providers that will store your data involved in using the service (e.g. your emails or photos)? Options: Yes; No; I don't know
16. What's your nationality?
17. What's your age?

4.2 Benchmark

As mentioned in the *2.3 The Demonstration Project*, the project is loosely based on an existing security camera product, Google's NestCam. The goal is to try to replicate its main features, but with the added benefits of the new architecture proposed above that decouples the service (camera hardware and software) from the infrastructure (cloud storage of the footage).

Figure 9: NestCam Indoor hardware

NestCam comes with the Wifi-enabled camera (Figure 9) and a smartphone application (Figure 10). The application can be used to view the live video-audio feed or to replay recording from an earlier time.

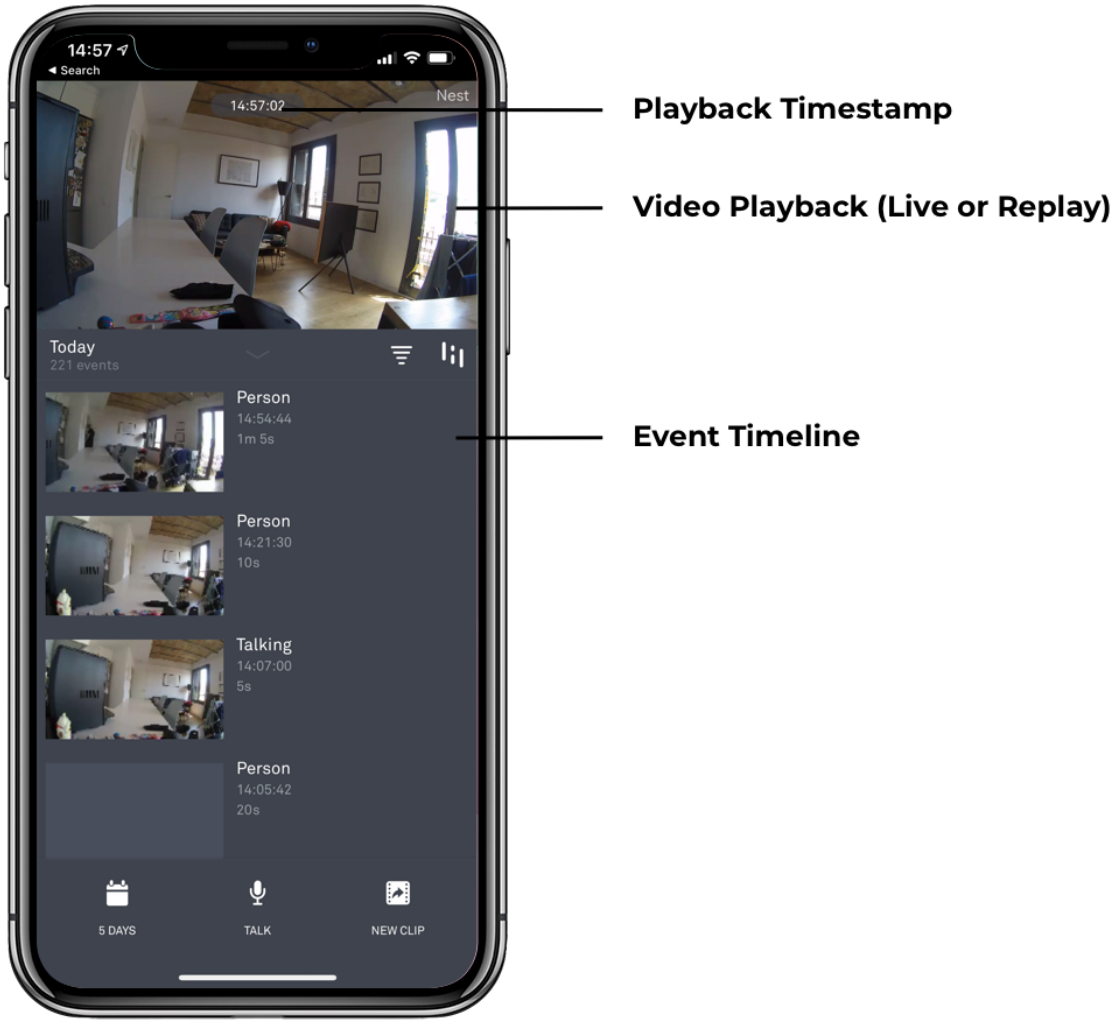


Figure 10: NestCam application user interface

NestCam comes with some advanced features that go beyond the scope of the prototype project. I list the features I decided to implement in my project compared to those of NestCam.

Feature	NestCam	Prototype
Watching live video+audio feed	✔	✔
Replaying recorded videos at an arbitrary time, up to 30 days	✔	✔
Event timeline showing still images from the past	✔	✔
Event push notifications (e.g. movement, person recognized)	✔	✘
Live talk to the camera speaker from the phone	✔	✘
Choice of storage of personal data	✘	✔
End-to-end encryption	✘	✔

4.3 Hardware

In order to build the a functioning security camera prototype, I had to obtain some hardware, namely:

- Computer to process signals from the camera and sync to the cloud
- Camera sensor and lens

In the following sections I will give an overview of the criteria I considered for both parts and the decisions have been made.

4.3.1 Computer

The computer had to be relatively cheap, small, with low energy consumption but powerful enough to do video processing and cloud syncing. It had to have the capacity to connect to the network preferably through WiFi and it had to support the camera hardware, of course. While I briefly considered building the prototype using the Arduino microcontroller board [25] and compatible hardware components, after a short investigation I concluded that it is too low-level to be practical for this project. I ended up choosing a Raspberry Pi 4 [26] computer with 4GB RAM (Figure 11). Raspberry Pi has become a de-facto standard hobby computer for hobby computing projects in the past few years with a lot of resources made available by the community online. It also fits the price, size and energy consumption criteria. I ended up buying a quick starter kit from Labists with all the necessary accessories (memory card, cables, case, etc) for 109.99 EUR.

4.3.2 Camera hardware

First of all, the camera hardware needed to be compatible with the computer that is meant to process its signals. Initially, I purchased a camera module called OV5647 with 5MP resolution, a 175° FOV and infrared sensor because its features made it attractive as a security camera module. However, it turned out to be incompatible with the computer case that came with the Raspberry Pi, so I ended up buying the standard Raspberry Pi Camera Module V2 8MP. Then, I ended up buying it again, after breaking the first one. The cost of one unit was 22.83 EUR.

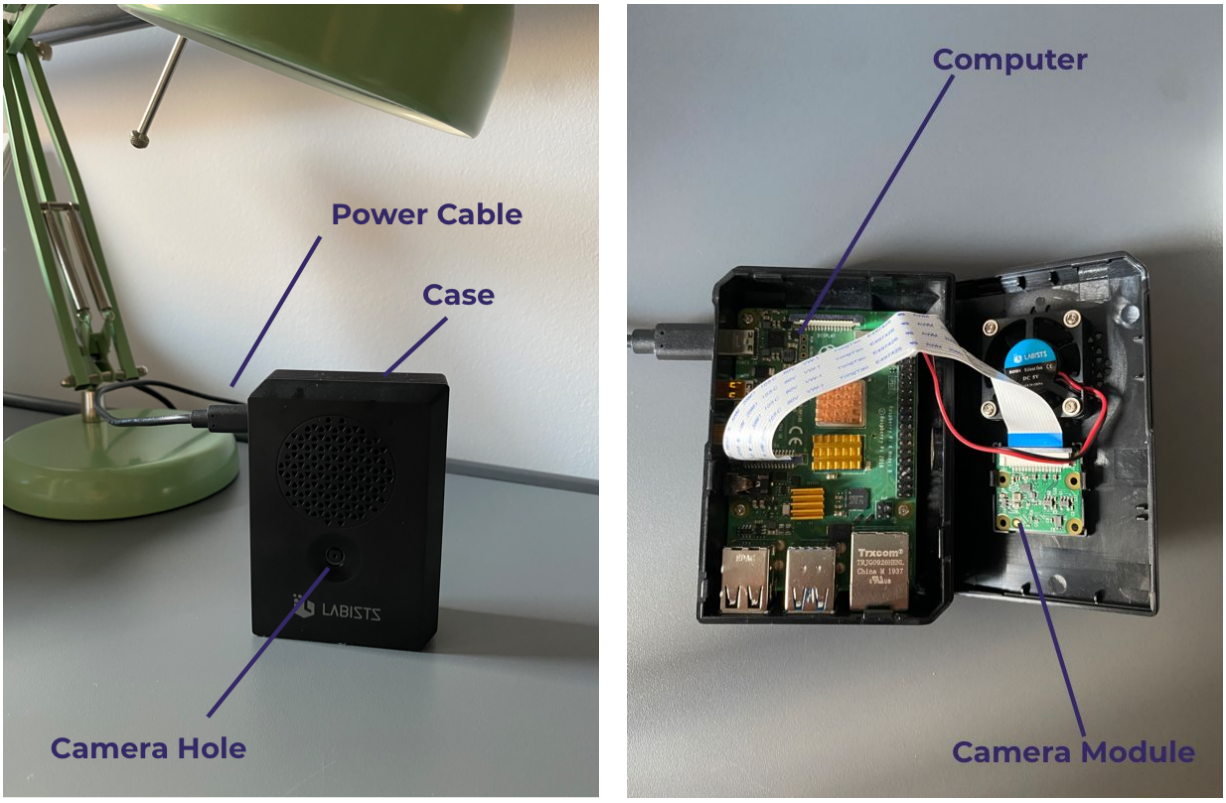


Figure 11: The computer and the camera module assembled in the case.

4.4 Software

The core of this project is the software that ties the system together and implements the features listed in the Benchmark section.

4.4.1 Components

The system consists of several components each responsible for a distinct task. Figure 12 shows a high level overview.

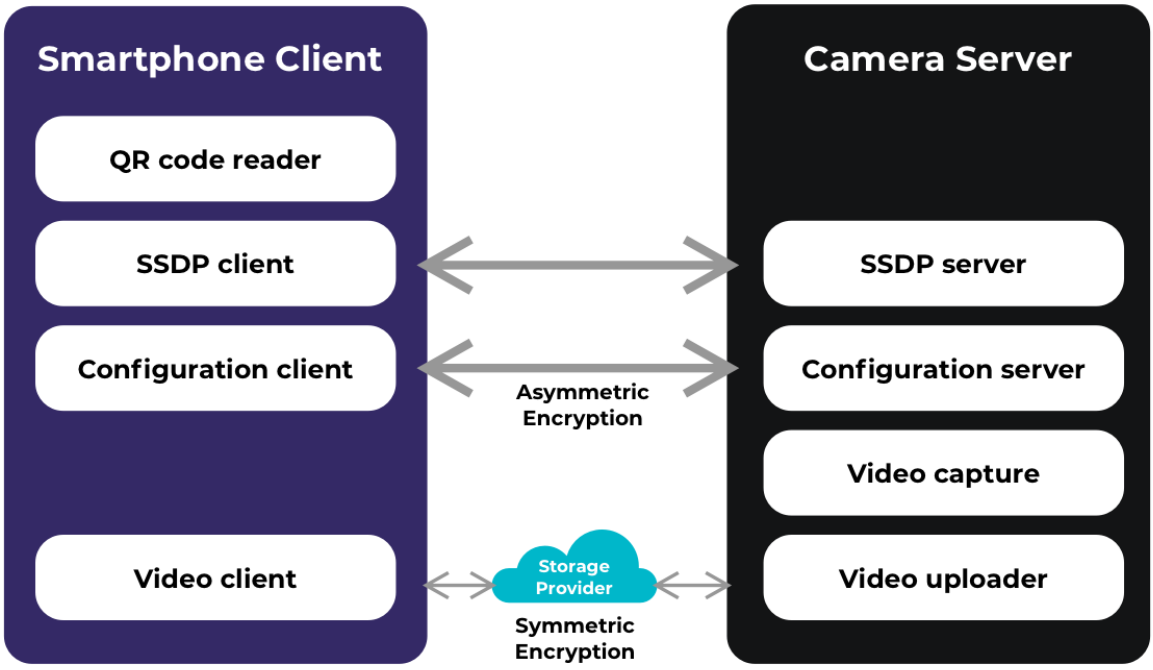


Figure 12: High level overview of the smartphone client and camera server components

4.4.1.1 QR code reader

This component is used during the system's initial configuration. It uses the built in camera on the smartphone to read a QR code glued on the case of the camera. The QR code encodes an RSA public key associated with the camera. This public key will be used to encrypt subsequent communication between the camera and the smartphone.

4.4.1.2 SSDP client, SSDP server

Once the public key of the camera has been read, the client has to find the camera's IP address on the network to proceed with the configuration. SSDP (Simple Service Discovery Protocol) is a protocol that takes advantage of broadcast IP addresses to advertise a service to potential clients connected to the same network. The prototype assumes that the camera server can connect to the network at least initially without any configuration, e.g. by using a cable. The SSDP server can be shut down after the configuration has completed.

4.4.1.3 Configuration client, Configuration server

Once the client finds the IP address of the camera server, it can proceed with the configuration. This means uploading the credentials of the storage provider obtained from the user as well as a random encryption key that will encrypt the video stream. This is done through a simple HTTP web server. To keep the connection secure, the client encrypts its requests with the public key of the camera. The camera signs responses using the private key so that the client can verify that those are authentic using the public key. No encryption is needed for the response, since there is no sensitive information passed back to the client. The Configuration server can be shut down after the configuration has completed.

4.4.1.4 Video capture

The camera needs to run a software that captures video signals from the camera module, encrypts, compresses, and stores them on the device until they can be synchronized.

4.4.1.5 Video client, Video uploader

The video files are encrypted with the symmetric key exchanged during the configuration phase and uploaded to the storage provider. The client in turn can obtain these from the same provider. This is the gist of the project; the storage provider is abstracted from the entire system and can be chosen by the user.

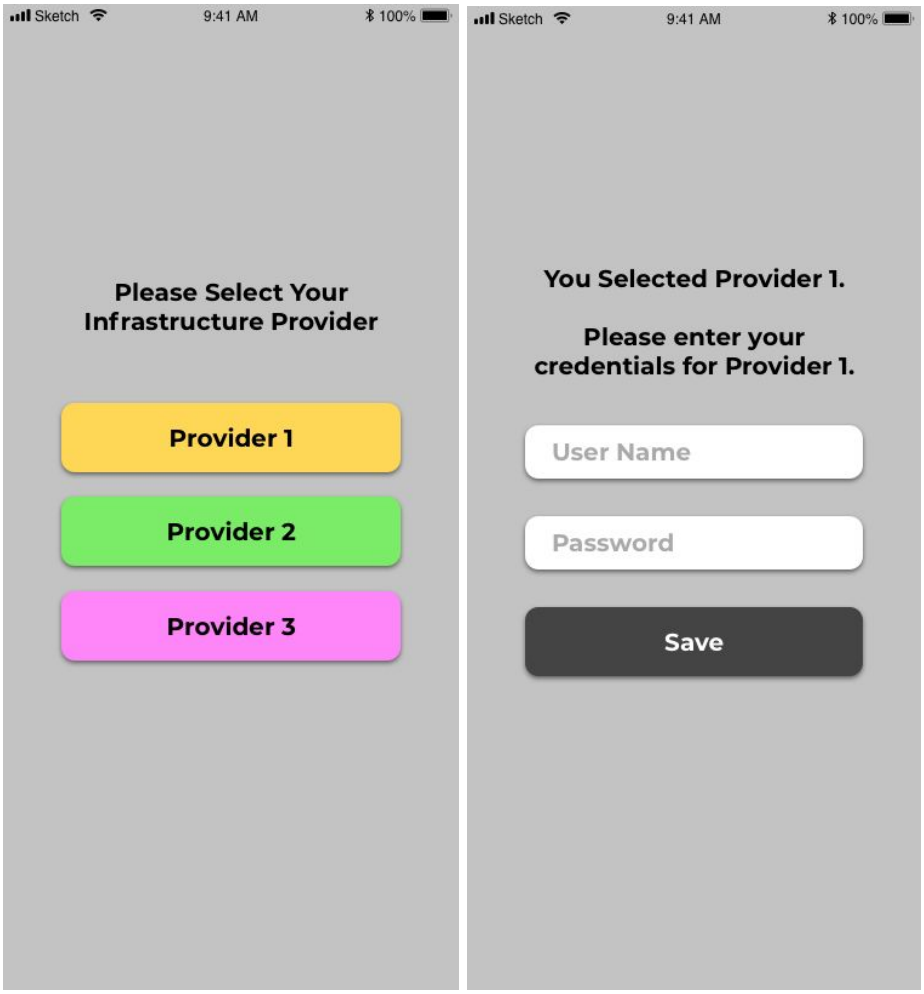
4.4.2 User Interface

As mentioned in the *4.2 Benchmark* section, I used the NestCam application as a benchmark for the set of features as well as inspiration for the user interface. I used a prototyping software called Sketch [27] to draw the following mockups prior to coding.

4.4.2.1 Read QR code

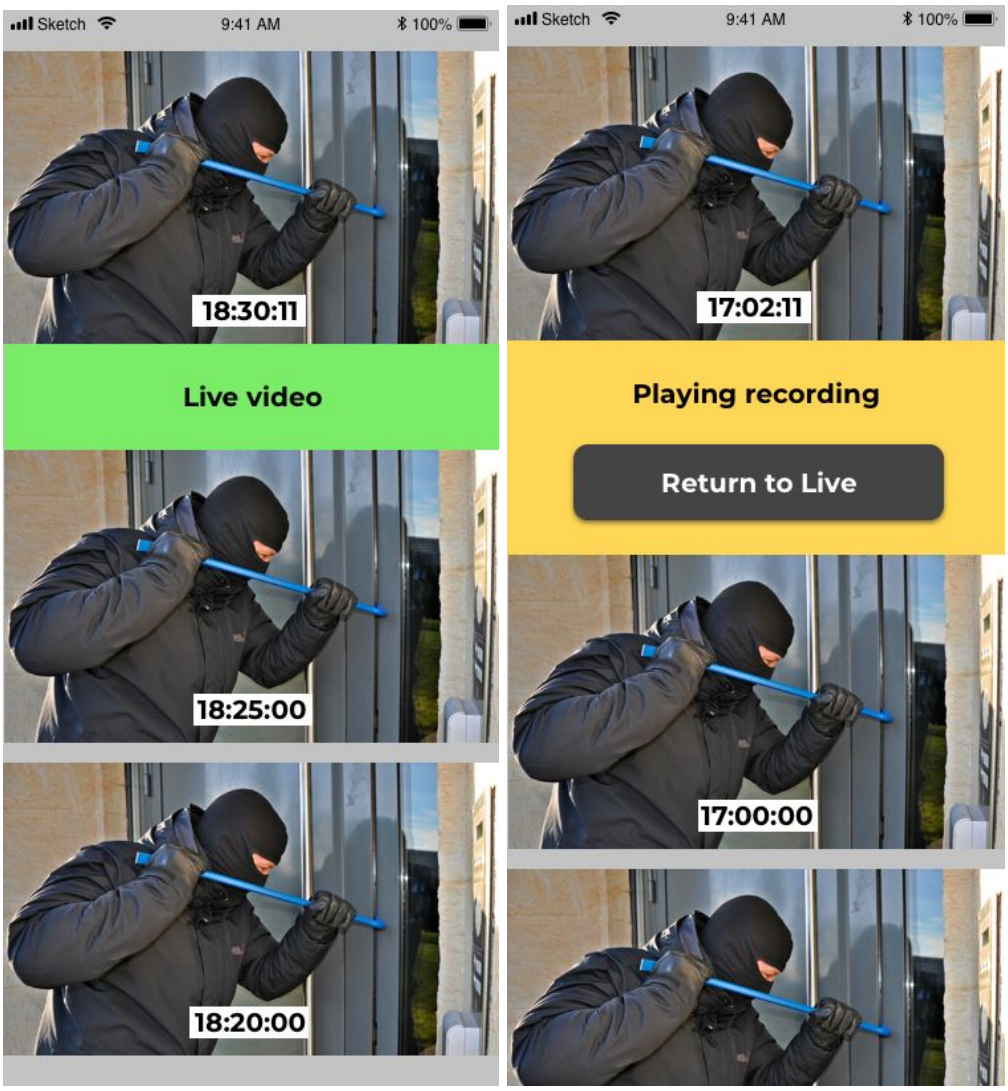
This is a simple screen consisting of only a title and a camera viewfinder. Pointing the smartphone's camera to the QR code containing the public key of the camera will automatically scan and validate the code and take the user to the next screen.

4.4.2.2 Select Storage Provider



This screen is the essence of the entire project. This is where all the innovation lies. On this screen the user can choose from a list of infrastructure-as-a-service providers such as Amazon AWS, Digital Ocean, Microsoft Azure, etc. After the user selected the provider from a predefined list, they can enter the credentials that belong to that provider.

4.4.2.3 Video view

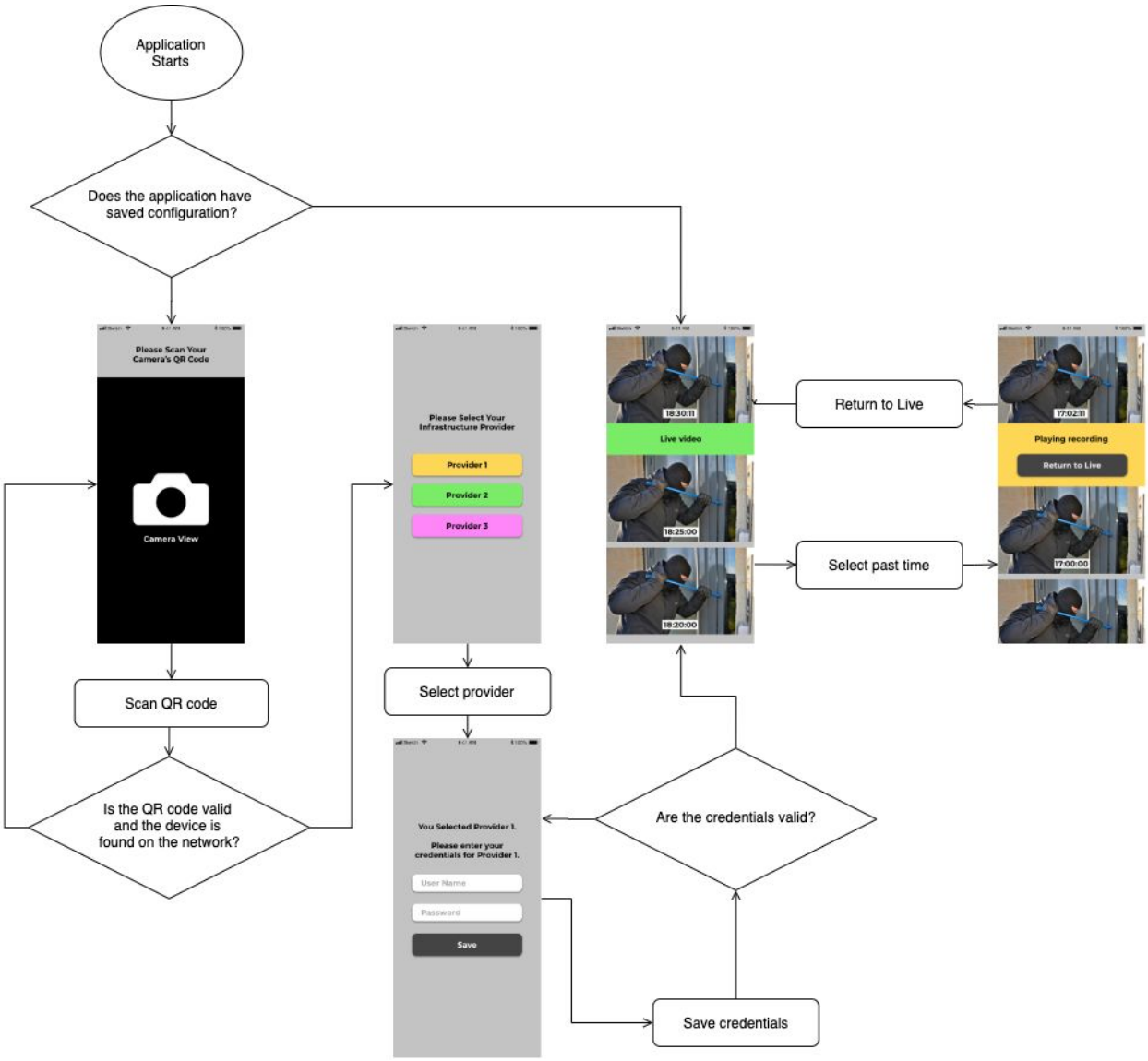


Tireless burglar photo provided by Pixabay.com

After the configuration is complete, this is the only functional screen of the application. On the top part it shows a video playback with a moving picture. It's either a live feed from the camera device (default) or, if the user selects a frame from the list of ordered past frames on the bottom part of the screen, it starts showing a playback of that recording. In playback mode the user can return to live view using the "Return to Live" button.

4.4.2.4 Storyboard

Below is a storyboard that shows the flow of interaction between screens.



4.4.3 Technologies

Raspberry Pi comes with a special Linux distribution as its operating system, called Raspberry Pi OS (formerly known as Raspbian). It is based on the Debian Linux distribution. Therefore it was important that the syncing server is compatible with Linux. This sets compatibility constraints on the server software. Regarding the smartphone client, the main target is iOS, but it is my preference to make the client available for Android as well.

4.4.3.1 Camera Server

After investigating several options, I chose Python3 to implement the bulk of the server software. Python has available open-source libraries to serve web requests, cryptographic functions, implement an SSDP server and most importantly, the storage providers I considered have client libraries written in Python.

For the video capture component I chose to work with FFMPEG [28], which is the de-facto standard multi-platform video and audio conversion toolkit and it's extremely powerful. I integrated FFMPEG with my Python code using subprocesses.

4.4.3.2 Smartphone Client

The choice here was between using a native development software development kit and language (e.g. Swift or Objective C) or using a cross-platform framework that produces application binaries for both major smartphone platforms (Android, iOS). After investigating available tools and libraries, I chose to work with React Native [29], which is an open source cross-platform framework based on React and Javascript, maintained by Facebook. This choice had some drawbacks deep into the development because some of the libraries were not as high quality as I expected. Furthermore, Javascript doesn't have proper types for binary strings (only UTF-16) [30], which caused some challenges in the cryptographic modules.

4.5 Development Methodology

As students, here we might be expected to refer to some decades-old development methodology we read about in CO1108, like waterfall-style Design, Implementation, Testing, etc, but for a team of one developer it's not practical to use an overly rigid lifecycle like that. When working on the project I took many ideas from Extreme Programming (XP) that has been adapted for single-developer teams [31] :

- Working with an ordered list of feature sets, taking the top, implementing, reordering
- Doing small releases
- Frequent refactoring
- On-site customer - "If you are your own customer then mumbling to ones self is reasonable."
- Coding standards

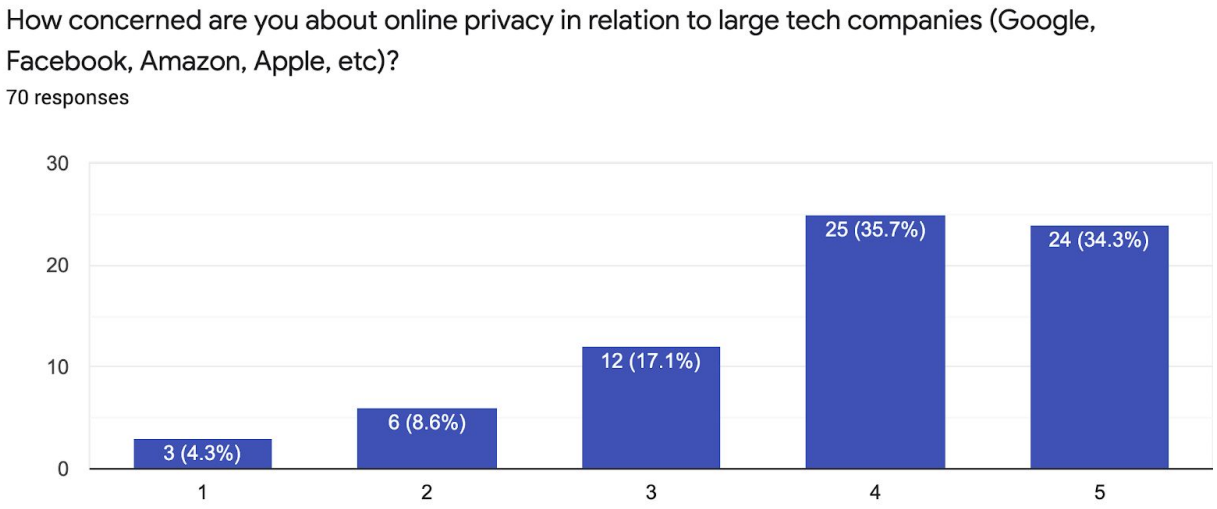
I used Git [32] as a version control system to keep track of changes in my code in both code bases (client and server). I used PhpStorm [33] and PyCharm [34] as my main IDEs, with occasional work in XCode [35].

5 Results

5.1 Survey

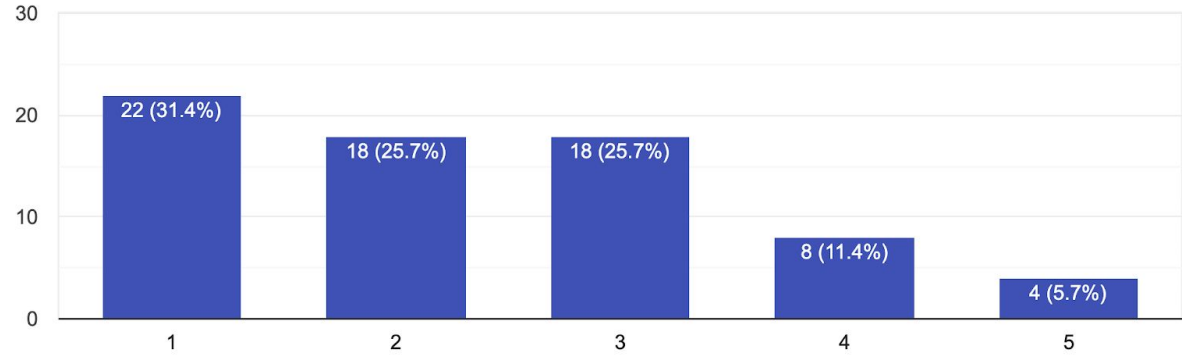
As described in the *4.1 Who cares?* section, prior to committing to the subject of the project, I decided to use an online survey to verify the existence of the problem: that is whether people are indeed concerned about their personal data stored and handled by tech giants; and also to validate my proposal: whether they would be more at peace if they could choose where their data is stored and had full control and visibility over it.

In the *4.1 Who cares?* section I presented the thoughts that went into selecting the audience and questions as well as the methods of obtaining the answers and their statistical significance. In this section I'm presenting the results, with my commentary and interpretation.



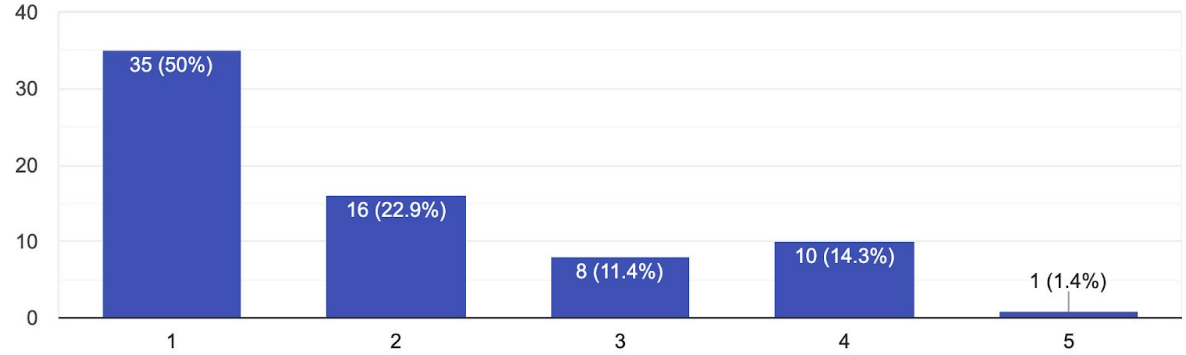
How much do you trust large tech companies (Google, Facebook, Amazon, Apple, etc) to handle your private data responsibly?

70 responses



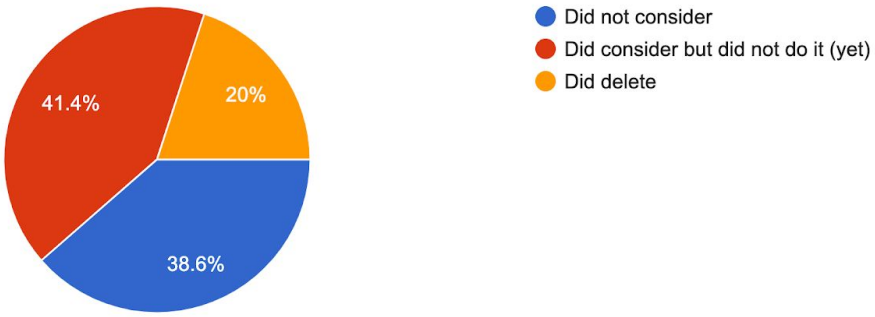
How much do you trust large tech companies to completely erase all your personal data when you remove your account?

70 responses



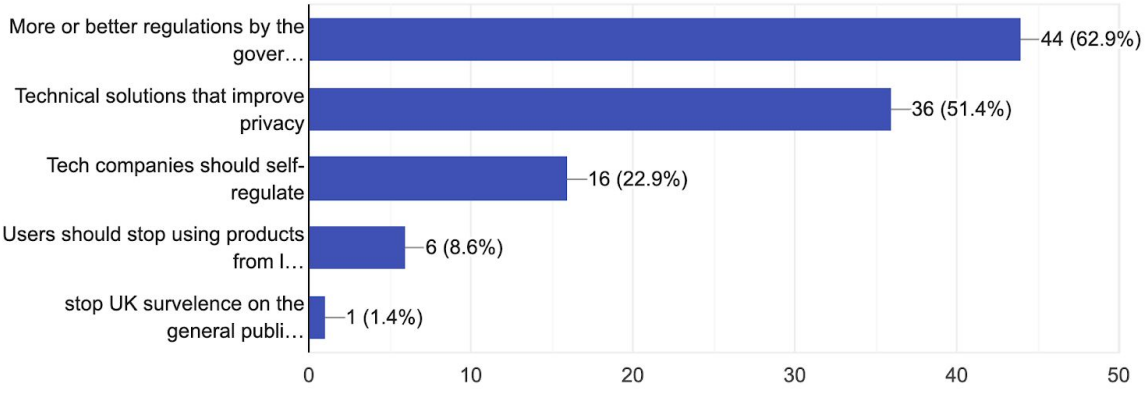
Have you ever deleted or considered deleting any of your social media or cloud storage (email, files) accounts due to privacy concerns?

70 responses



What steps do you think need to be taken to improve online privacy in relation to large tech companies?

70 responses

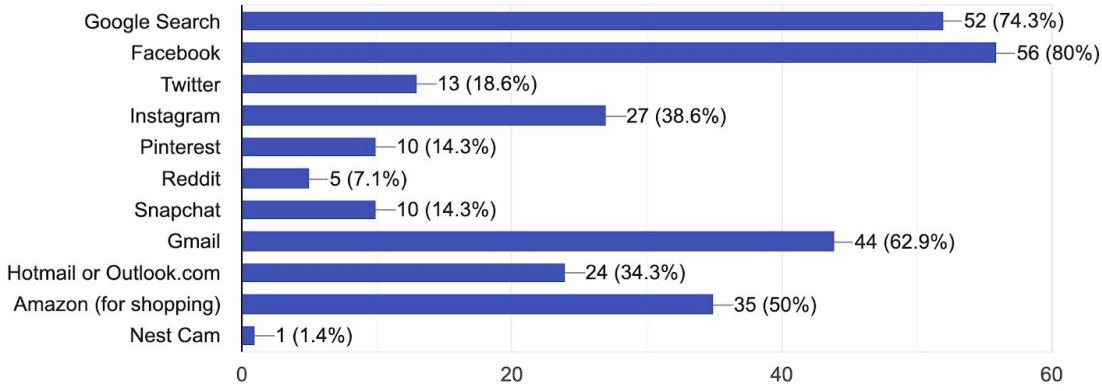


From the answers to the first 5 questions above it's clear that there's a trust issue between the tech giants like Google, Facebook, Amazon, Apple, etc. and their (potential) users. Perhaps the most striking results is the one with the data-deletion question: Half of the respondents don't believe at all that their data is fully removed when they close their account with a tech company.

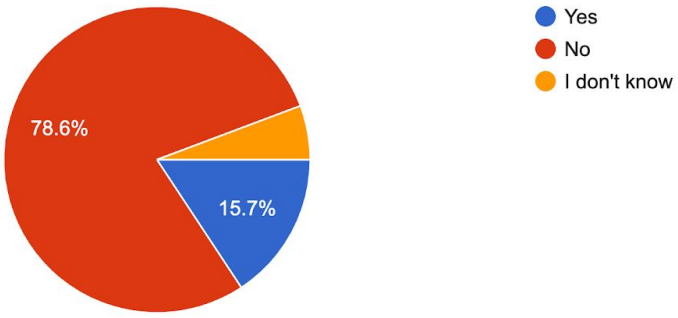
The majority at least considered or already deleted some accounts due to privacy concerns, which verifies that the problem is real.

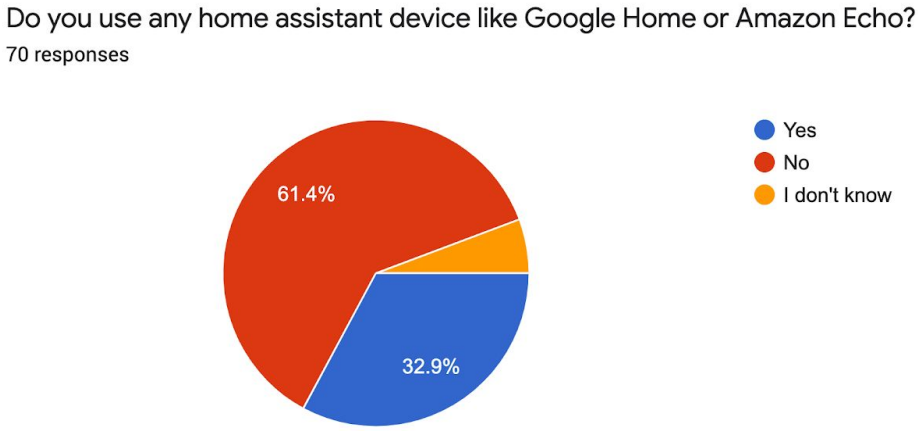
Most people seem to think that the solutions to privacy issues should come from the government or some sort of technical solution, oer perhaps a combination of both.

Which of these services are you using actively?
70 responses



Do you own any home security device (e.g. camera) that uploads data to the cloud (e.g. a video feed)?
70 responses



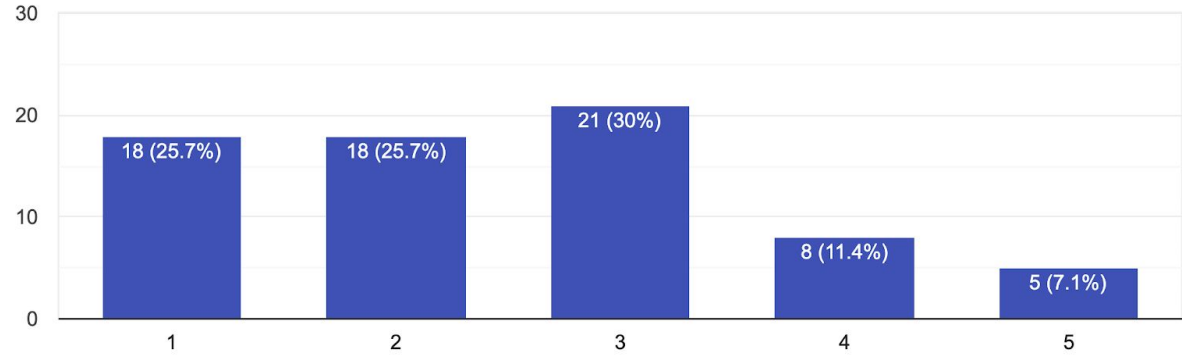


Based on responses to the 3 questions above, only 1% of the respondents own a NestCam, and 16% of them use some sort of cloud-connected home security device. Further, approximately one third of the respondents own an internet-connected home assistant, which poses similar privacy concerns as a security camera. From the point of view of my project is meant that the prototype project won't solve a mass problem (although 16% is significant), but the concept of separating infrastructure from the service is generally applicable to a broader set of services, some of which are much more popular among the respondents (Facebook, Gmail, Instagram, …).

It can be said that the respondents use a broad set of cloud services, but it's important to point out that the responses here are skewed by the collection method - namely Facebook campaigns, which run on Facebook and Instagram, which explains why only 80% of the respondents indicated that they use Facebook.

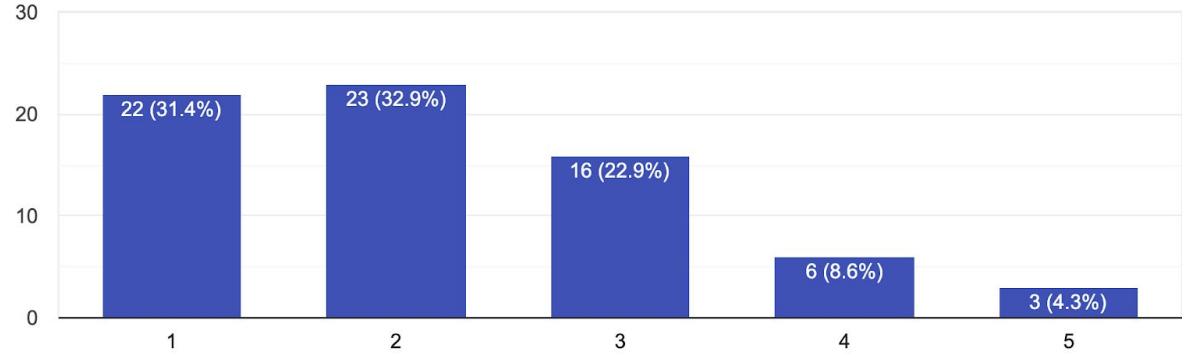
How complete idea you have about what data is collected and kept about you by the large tech companies?

70 responses

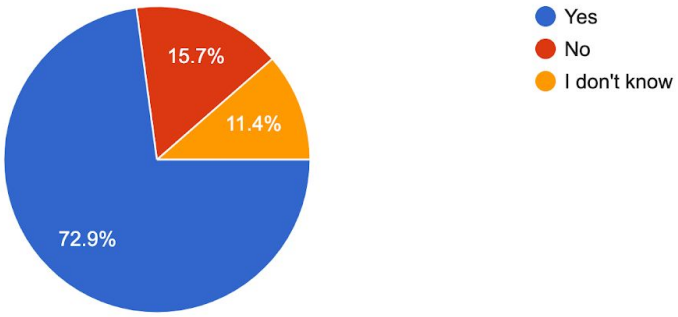


How complete idea you have about where and how the data collected about you stored by the large tech companies?

70 responses

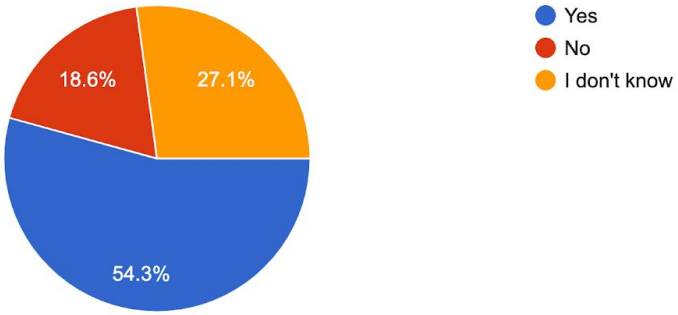


Would you be interested in seeing the complete raw data stored about you by large tech companies?
70 responses



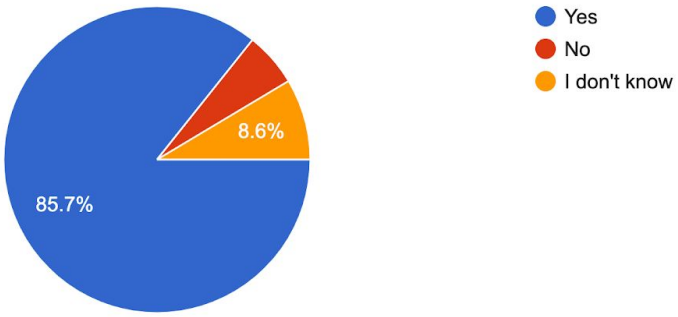
The responses to the 3 questions above indicate that respondents generally have an incomplete idea of what sort of personal data is stored about them by large tech companies and how these are collected, even though a convincing majority of them would be interested to see this data.

Would you feel more comfortable using an online service if you could select where (in which country) your data is stored?
70 responses



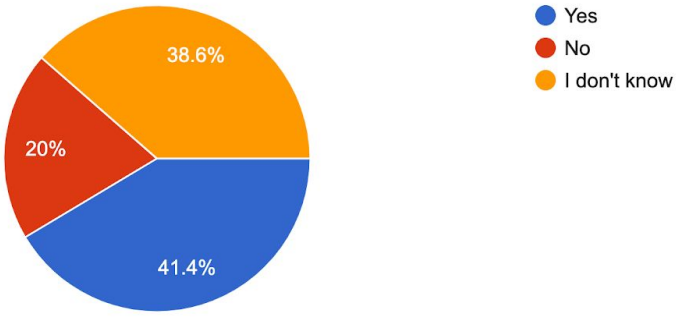
Would you feel more comfortable using an online service if you could fully see your stored data at all times?

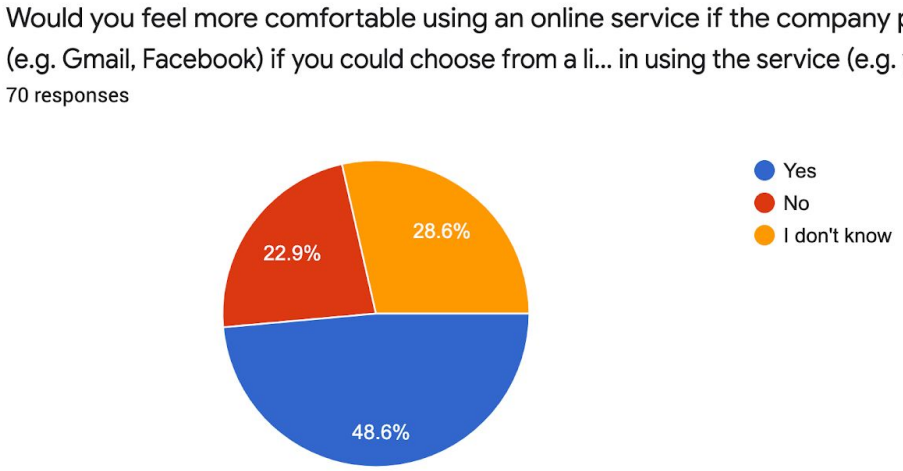
70 responses



Would you feel more comfortable using an online service if the company providing the service (e.g. Gmail, Facebook) was different from the one s... in using the service (e.g. your emails or photos)?

70 responses





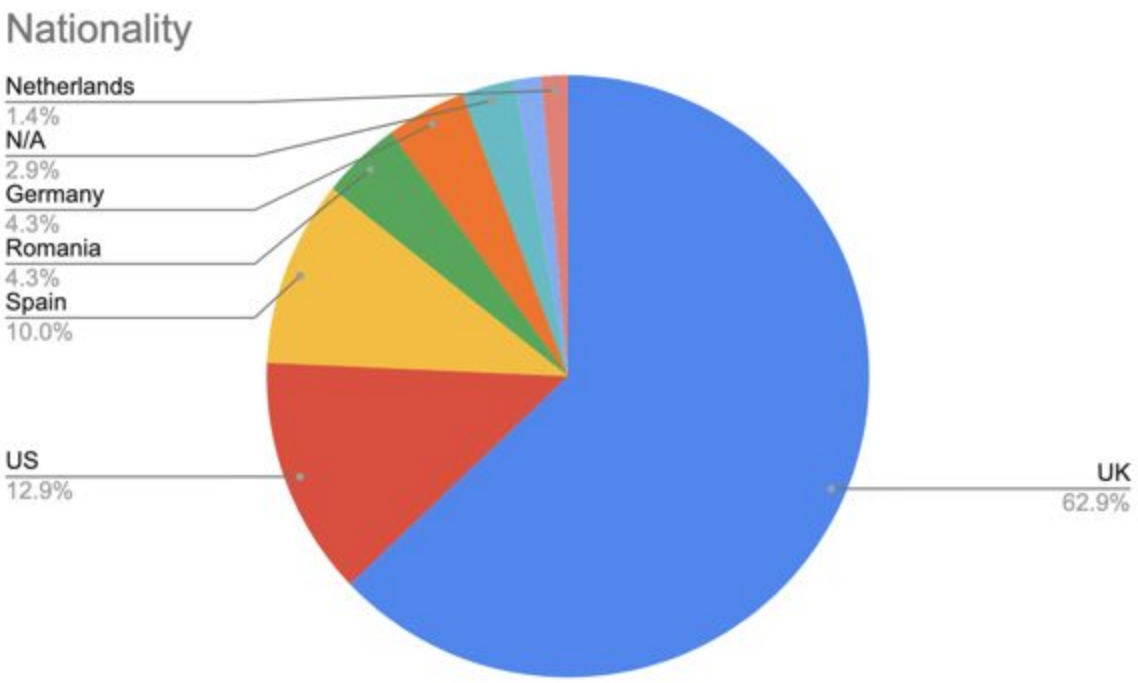
The responses to the 4 last questions were meant to validate my proposed solution: that it would indeed increase the comfort of users of cloud-connected services. The most striking set of responses were given to the question related to the visibility of the data. A very convincing majority of 87% would feel more comfortable if they could see the actual data being stored about them. This is certainly one of the advantages of the proposed architecture: full visibility to the entire set of data.

However, visibility doesn't mean interpretability, which is important to keep in mind. The access to, say, a large set of text files in a proprietary format doesn't increase interpretability. Nevertheless, full ownership of and access to the data might create a market for companies specializing in interpreting and visualizing the personal data stored on the chosen infrastructure provider. One can imagine a company that connects to their choice of storage, pulls all the data stored there by Facebook and creates beautiful visualizations and easy-to-use tools that help the user understand and explore it. I expand on that in *6.2 Data portability and interoperability*.

As to the questions regarding separating the service provider from the infrastructure provider and offering a variety of choices for the infrastructure provider, the answers aren't so convincing, although the plurality still prefers these solutions. It might take some explaining and education to convey the advantages (and disadvantages) of such architecture.

Nevertheless, the results above convinced me that my proposed solution is valid and a large number of people would find it valuable.

5.1.1 Demographics



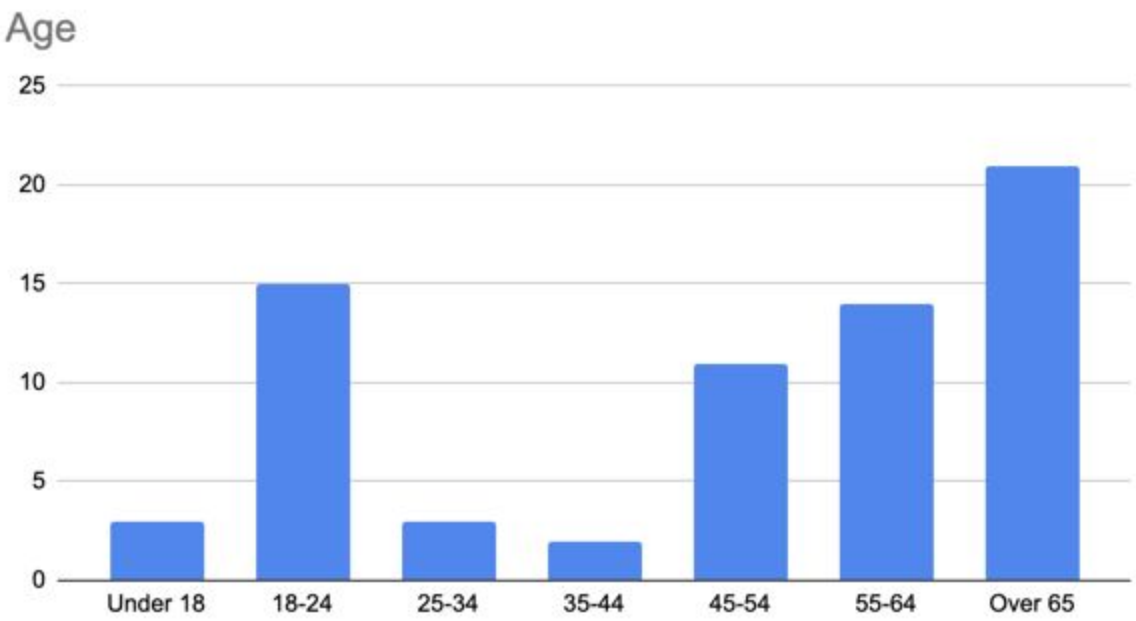
Since the nationality field has free text input, I had to normalize it. As mentioned in the *4.1 Who cares?* section, I targeted the United Kingdom, United States and Spain. A small number of expats show up among the respondents.

Even though the Facebook campaign has achieved much more impressions and clicks from Spain than the other two targeted territories (see Figure 13), the overwhelming majority of the respondents were British. This could perhaps be explained by the language barrier.

Name	Impressions	Unique Link Clicks
UK	13,570	1,031
ES	83,628	4,830
US	2,985	131
Results from 3 ad sets	100,183 Total	5,990 Total

Figure 13: Facebook campaign impression and click count per territory

In terms of the age of the respondents, older people (55+) were much more likely to respond. There's also a significant number of people between 18-24, who perhaps responded to the message of the advertisement better ("help me graduate").



Interestingly enough, when segmenting the answers for age (e.g. excluding people 55 or older) the summary of the responses to the privacy and storage related questions (e.g. *"Would you feel more comfortable using an online service if the company providing the service (e.g. Gmail,*

Facebook) was different from the one storing your personal data involved in using the service (e.g. your emails or photos)?") do not differ between age groups significantly.

I include the raw set of responses in the Appendix under *9.3 Survey answers*.

5.2 Prototype

This section describes the implementation of the security camera prototype in detail, built around the idea of the choice of infrastructure provider.

5.2.1 Branding

Applications cannot exist without branding; a name and an icon are bare minimum requirements. Software projects typically require a name or at least a code name to refer to as well (e.g. package names, repository names, etc).

I christened the prototype **Choice Cam**. It refers to the core concept of the project: the choice over where and how the user's data is stored. I also registered the choicecam.org domain name where I plan to publish the source code and setup instructions to anyone who is interested.

As for the application icon and logo, I employed my incredible design talent to come up with the visual branding (Figure 14).

Figure 14: Choice Cam logo and application icon on an iPhone home screen

The three circles symbolize radio buttons (standard computer form elements for multiple choice options) representing choice, while the camera lens is a visual pun symbolizing both a selected radio button as well as a security camera. The bright yellow color is meant to divert the user's attention from the lacking features of the application.

5.2.2 HTTP Live Streaming

When investigating technologies to use to stream live video, I came across HTTP Live Streaming (HLS) [36]. It's one of the most widely used [37] and supported streaming protocols, designed by Apple and built on top of existing technologies. It's typically used to stream live or on-demand video.

Some of its features made it an ideal choice for this project:

- Built on HTTP. Unlike some other video streaming protocols using UDP-based protocols such as Real-time Transport Protocol, HLS uses plain old HTTP requests. This is an extremely valuable attribute from the perspective of this project, because the storage providers I considered support HTTP access to the stored data out of the box but might not support other protocols.
- Supports encrypted video. Since the idea is to host the video over HTTP on the selected storage provider, some type of authentication or encryption is crucial. HLS supports the encryption and decryption of the video feed with AES-128 symmetric encryption algorithm, which makes it possible to end-to-end encrypt the video from the camera server to the smartphone client. This means that the storage provider has no access to the contents.
- Native support on Android and iOS. Since part of the prototype is a smartphone application, it was extremely important to choose a technology that has native support on the major smartphone platforms.
- Built on top of common video technologies. HLS uses H.264 [38] video encoding and MPEG-2 TS video container format, both are commonly used technologies with a wide range of tools to perform encoding and playback. To implement live streaming, it relies on extensions over the M3U file format, another widely used and time-tested format that has been around since the Winamp days.

Based on the above, HLS seemed to be the obvious choice of technology for the video streaming part of the prototype. How does it work?

The core of the protocol is the playlist file. It's a plain-text file containing a list of video files called segments, each a few seconds long. The live video (and audio) data is continuously recorded, and new segments are added to the list every few seconds or so.

Here's an example HLS playlist file.

```
#EXTM3U
#EXT-X-TARGETDURATION:5
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:1
#EXTINF:4.8,
video-segment-1.ts
#EXTINF:4.9,
video-segment-2.ts
#EXTINF:5.0,
video-segment-3.ts
```

The playlist file contains 3 video segments and their corresponding URLs, each approximately 5 seconds long. The actual duration is indicated in the `EXTINF` tag above each segment. The lack of `EXT-X-ENDLIST` tag in the end of the file tells the client that it should expect more segments to be appended, and so the client continuously polls the playlist URL to receive the new segments added to the end of the list. In case of live streaming (as opposed to on-demand or event streaming) the playlist is updated using a sliding window fashion, keeping a constant number of segments as it discards old segments when new ones are added to the list. To help the client keep track of these changes, the so-called sequence number of the first segment in the list is indicated in the `EXT-X-MEDIA-SEQUENCE` tag.

Here's how the file above would look when updated given a sliding window:

```
#EXTM3U
#EXT-X-TARGETDURATION:5
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:2
#EXTINF:4.8,
video-segment-1.ts
#EXTINF:4.9,
video-segment-2.ts
#EXTINF:5.0,
Video-segment-3.ts
#EXTINF:4.7,
video-segment-4.ts
```

HLS supports encoded segment files with AES-128 [39] encoding. The key has to be accessible at a URL indicated in the playlist file like so:

```
#EXT-X-KEY:METHOD=AES-128,URI=path/to/key
```

If encryption is used, the client expects encrypted segment files and automatically decrypts them using the key prior to playback. The IV (initialization vector) is either indicated in the `EXT-X-KEY` tag or is contained in the segment file itself.

While the camera server of Choice Cam does create a HLS playlist when encoding the live video feed, the smartphone client does not use this. Instead, it constructs its own playlist file using a list of segments and other metadata. See *5.2.3.12 Video View* section.

5.2.3 Code review

Here we get to the juicy parts. In the following sections I'll present some key pieces of code of the implementation. To understand the pieces of the puzzle and how they fit together, I recommend reviewing the architecture detailed in the *4.4 Software* subsection in the Methods section and watching the demonstration video attached in the auxiliary files (see *5.5. Demonstration*).

The full code is included in the Appendix (*9.4 Source code*), and also attached in full to the project in a separate ZIP file with running instructions. In the Appendix I also include a full list of software dependencies used by the prototype (*9.5 List of software dependencies*).

The code review enumerates the components of the camera server, and then covers the functions of the smartphone client.

5.2.3.1 Generating keys

This component of the server is not used during the interaction of the user with the prototype. If this was a real, marketable product, this code would be used by the manufacturer to generate a unique public-private key pair that is shipped with the camera, as well as a visual representation of the public key (QR code) that is printed and glued on the camera's packaging or case. For maximum security the manufacturer would have to make sure to ship the camera in a tamper-proof packaging.

The generated key pair will be used during the configuration phase when the smartphone client needs to "talk to" the camera server. It ensures that the storage provider credentials and the symmetric key used to encode the video (similar to a session key in TLS [40]) cannot be stolen by a man-in-the-middle attack when passed from the smartphone client to the camera server.

`generate_keys.py`

```
from Crypto.PublicKey import RSA
from Crypto import Random
import os
import qrcode
from config import PRIVATE_KEY_FILE_PATH, PUBLIC_KEY_FILE_PATH, PUBLIC_KEY_BARE_FILE_PATH,
PUBLIC_KEY_QR_CODE_FILE_PATH

if not os.path.isfile(PUBLIC_KEY_FILE_PATH) and \
  not os.path.isfile(PRIVATE_KEY_FILE_PATH) and \
  not os.path.isfile(PUBLIC_KEY_BARE_FILE_PATH) and \
```

```
not os.path.isfile(PUBLIC_KEY_QR_CODE_FILE_PATH):

 random_generator = Random.new().read
 key = RSA.generate(2048, random_generator)
 private, public = key.exportKey(), key.publickey().exportKey()
 bare_public = public \
     .replace(b"\n", b"") \
     .replace(b"-----BEGIN PUBLIC KEY-----", b"") \
     .replace(b"-----END PUBLIC KEY-----", b"")

 with open(PRIVATE_KEY_FILE_PATH, 'wb') as private_file:
     private_file.write(private)
 with open(PUBLIC_KEY_FILE_PATH, 'wb') as public_file:
     public_file.write(public)
 with open(PUBLIC_KEY_BARE_FILE_PATH, 'wb') as bare_public_file:
     bare_public_file.write(bare_public)

 image = qrcode.make(bare_public)
 image.save(PUBLIC_KEY_QR_CODE_FILE_PATH)
```

The code is using libraries pycrypto (2.6.1) [41] and qrcode (6.1) [42]. In the first few lines it checks if the keys already exist, in which case it doesn't do anything. On the contrary, it generates a random 2048 bit RSA public-private key pair using a secure random number generator. The United States National Institute of Standards and Technology currently recommends at least 2048 bit key sizes for RSA [43]. Naturally, the private key has to be protected on the device.

The public key is also saved in a different format ("bare"), stripping it from its PEM DER ASN.1 PKCS#1 [44] headers as well as wrapping new-line characters, leaving only the modulus and exponent in Base-64 encoding. We are left with a Base-64 encoded DER file. This is necessary for the SSDP server (see 5.2.3.2) that responds with this string as the Unique Service Name (USN) when the client is searching for it.

Finally, the bare public key is encoded onto a QR code (see Figure 15) as plain text and saved along with the keys. Note that the QR code could be made smaller by storing the DER file without the Base64 encoding, but this might result in issues with Javascript that lacks proper binary types [30].

Figure 15: The QR code corresponding to an example public key.

5.2.3.2 SSDP server

The role of the SSDP server is to help the smartphone client find the camera server on the network during configuration. SSDP (Simple Service Discovery Protocol) [45] is a protocol that uses special multicast and unicast based versions of HTTP. Servers respond to a multicast HTTP-like search request issued by the clients by returning their Unique Service Name (USN). For this project I chose the camera server's public key as USN, this way the client can easily select the right server if there are several camera servers on the same network (e.g. the user is setting up a cluster of cameras).

SSDP requests and responses look like this.

Request:

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1901
MAN: "ssdp:discover"
MX: 1
ST: ssdp:choicecam
```

Response:

```
NOTIFY * HTTP/1.1
HOST:239.255.255.250
NT:ssdp:choicecam
```

```
NTS:ssdp:alive
USN:MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAkU3r2cjXJqhPfeFZtiYF6
wUJtvq+Bg8RK/fbM65/hOWzKG2y3DtmvsS/qG96WyTU+bwEkqf/5/r6hwyp0GRKivOcHb
ayBmf/U0qA5y5bpzmk/K/QZgoFjfqDSbtZkfnGjOMyag6QKaNXyQMKeWPPdA+KSoWQxd7
4iy/uC0ICWQ0b71hIfNT2YNZSHGWUZQybDw8GYFjCCWUmzo17kjlqqdTzkVDKXc4SSXgA
rGkJz6QXFxVidvjElIWMotg4paXDTA+I14sUG6iJDinYxWdRKrcqJloKG6ZuI5Sh7MOrP
jfWtE/bnuJZqMEyf9qNtWXBFha6sjlSJfZa9w02KoylqQIDAQAB
```

In my case they are sent over UDP (the more common transport layer protocol for SSDP) but it could work just as well on TCP.

Let's see what we've got here: the client sends a "search" request to the multicast IP address of the network, including a "service type" (ST) header, which is `ssdp:choicecam` in our case. This message is sent to all devices on the network. A device that has an SSDP server listening on port 1901 will pick up this message and check the service type. If it matches its own service type, it responds with a notify response including an `ssdp:alive` NTS header, the service type and a Unique Service Name (USN) uniquely identifying the device. For Choice Cam, I decided to use the camera server's public key here, so that the client can simply check if the QR code it read matches the USN in the SSDP response.

Let's see the code:

ssdp.py

```
from ssdpy import SSDPServer
from config import PUBLIC_KEY_BARE_FILE_PATH

def run():
   with open(PUBLIC_KEY_BARE_FILE_PATH, 'r') as bare_public_file:
       public_key = bare_public_file.read()

   server = SSDPServer(public_key, device_type="ssdp:choicecam", port=1901)
   server.serve_forever()
```

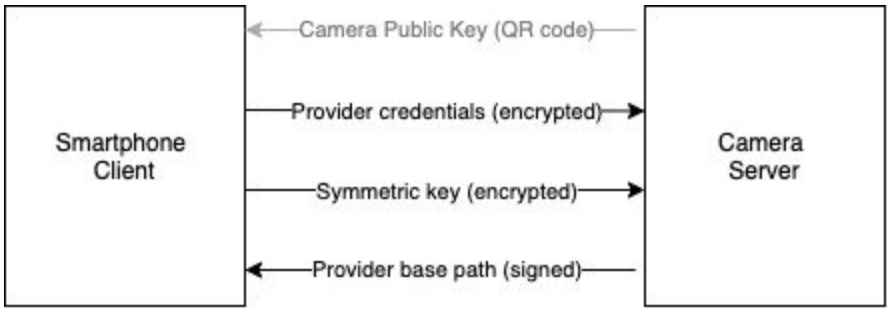
It uses the open-source library ssdpy (0.2.4) [46] which makes it very easy to set up and run the SSDP server. The code reads the public key saved in the disk (bare version, see *5.2.3.1 Generating keys*) and starts the SSDP server with the key as USN, and `ssdp:choicecam` as service type.

The `run()` routine is invoked by the application in a forked process as I explained in the `5.2.3.8 Run` section.

5.2.3.3 Configuration server

The next piece is the configuration server that receives the storage provider credentials and the symmetric key from the smartphone client. This is a simple web server. What's special about it is that it cannot rely on TLS [40] to secure the communication between the smartphone client and the camera server because TLS typically uses certificate authorities or self-signed certificates, none of those are workable in our case.

Instead, the communication is one-way encrypted in the Application layer (from client to server) and in turn unencrypted but signed the other way. It's because the server does not transmit sensitive information to the client but the authenticity of the response is important. For the encryption/signing process the camera server's public and private keys are used. The public key is read by the client as a QR code scan. After the configuration step is completed, the camera server does not communicate with the smartphone client directly, only through the storage provider.



Let's see the code.

webservice.py (partial)

```
def run():
    if not os.path.isfile(PRIVATE_KEY_FILE_PATH):
        raise RuntimeError('Server private key not found under ' + PRIVATE_KEY_FILE_PATH)

    socketserver.TCPServer.allow_reuse_address = True
    with socketserver.TCPServer(("", PORT), WebServiceHTTPRequestHandler) as httpd:
        try:
            httpd.serve_forever()
        except:
            httpd.server_close()
            raise
```

Here we make sure the private key file exists (to decrypt the message from the client) and start a TCP server with a custom handler handling incoming requests. Note that socketserver is a built-in Python module. Let's see the handler:

webservice.py (partial)

```
class WebServiceHTTPRequestHandler(http.server.BaseHTTPRequestHandler):

   private_key = None

   def do_POST(self):
       try:
           content_length = int(self.headers['Content-Length'])
           body = self.rfile.read(content_length)

           message = self.decrypt_body(body)
           provider = provider_factory(message['provider_data'])

           if not provider.credentials_correct():
               self.send_signed_response(400, b"Incorrect credentials")
               return

           save_config_for_streaming(message)

           base_path = provider.base_path()
           self.send_signed_response(200, base_path.encode())

           # Configuration no longer needed, shutting down server.
           sys.exit(0)
       except Exception as e:
           print(e, file=sys.stderr)
           traceback.print_exc()
           self.send_signed_response(500, b"Unknown error")
```

The handler only implements the `do_POST` method as it doesn't require to handle any other HTTP methods. It doesn't check the URI since there's only one possible operation. It simply reads the raw request body sent from the client and encoded with the public key and decrypts it using the private key. Then, it attempts to create a provider object based on the data from the client. I will talk in detail about the provider objects in the *5.2.3.4 Providers* section. It then checks that the credentials passed by the clients work and if so, it saves the configuration to the server and returns a signed response with the base path of the video files.

Finally, it shuts down for security reasons - malicious actors should not try to reconfigure the server unless it's reset through physical access. It's because someone at the factory could have

read the public key and by connecting to the same network (e.g. Wifi connection from another apartment) could in theory gain access to the video feed.

The `decrypt_body` and `send_signed_response` methods can be found in the full source code in the Appendix (*9.4.1 Camera server*). I omit them here for brevity and because there is nothing surprising about them.

5.2.3.4 Providers

Providers are the heart and soul of the prototype. They provide the abstraction over infrastructure that is the core idea of the project. The user has the choice where their data is stored from the very beginning.

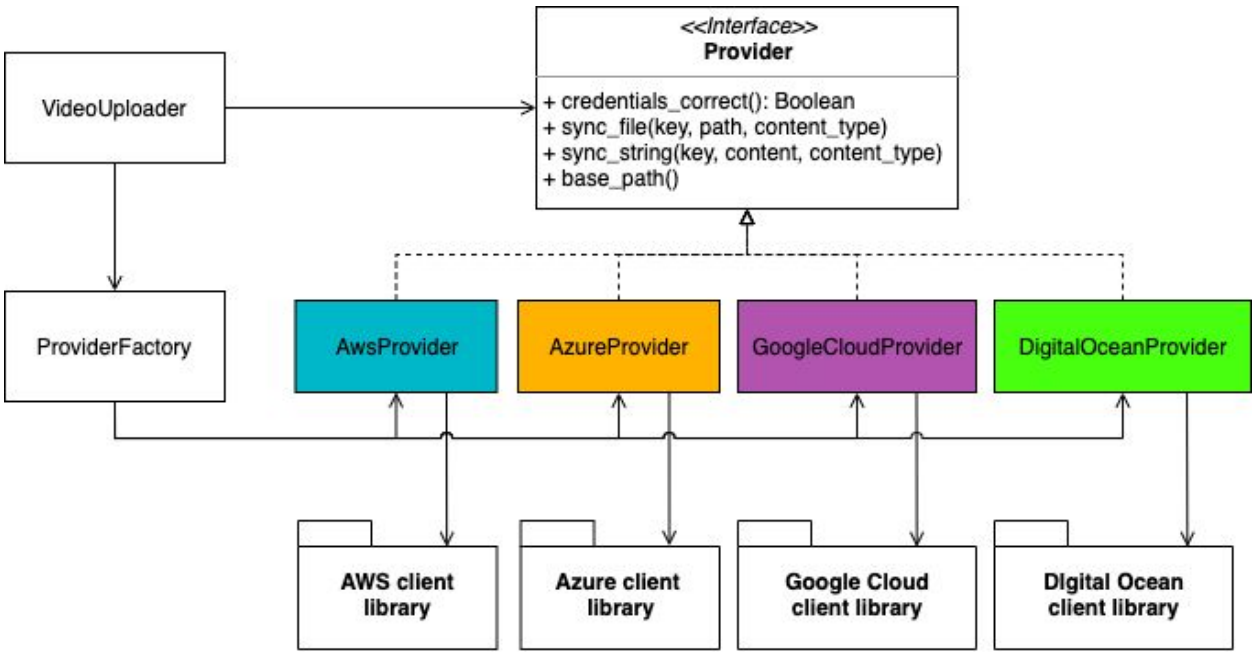


Figure 16: UML diagram of the providers. Note that Python doesn't have formal interfaces like other languages (e.g. Java, C#) so the interface on the UML diagram is merely conceptual.

The implementation infrastructure (in our case: storage) providers rely on two important Object-Oriented Programming (OOP) concepts: Abstraction and Polymorphism (Figure 16). The client don't doesn't have to know how the video files are uploaded to the storage provider (Abstraction), in fact it doesn't even need to know which provider is used (Polymorphism).

Each provider class is needs to perform 4 functions:

1. Check if the credentials provided at initialization are correct, e.g. by trying to perform an operation through the client library.
2. Upload (sync) a local file to a particular path (or key) to the remote storage.
3. Upload (sync) a string to a particular path (or key) to the remote storage.
4. Return the HTTP base path where the uploaded files will be available to download.

About the last point: as mentioned in the *5.2.2 HTTP Live Streaming* section, the smartphone client will use regular old HTTP requests to an otherwise open-access web server to download the encrypted video and image files. The base path here consists of the protocol, host name and path of that web server.

Let's look at a concrete provider implementation, the one using Amazon Web Service's S3 object storage product as its backend.

`providers/AwsProvider.py`

```
import boto3
from botocore.client import Config

class AwsProvider:
    def __init__(self, credentials):
        self.bucket_name = credentials['bucket_name']
        self.region = credentials['region']
        self.key = credentials['key']
        self.secret = credentials['secret']
        self.client = None

        pass

    def credentials_correct(self):
        try:
            client = self.get_client()
            client.list_objects(Bucket=self.bucket_name)
        except:
            return False

        return True

    def sync_file(self, key, path, content_type):
        print("Uploading " + path + " to " + key)

        with open(path, 'rb') as data:
            self.get_client().put_object(Key=key, Body=data, Bucket=self.bucket_name,
                                         ACL='public-read',
```

```
                                StorageClass='INTELLIGENT_TIERING',
ContentType=content_type)

    def sync_string(self, key, content, content_type):
        self.get_client().put_object(Key=key, Body=content, Bucket=self.bucket_name,
                                ACL='public-read',
                                StorageClass='INTELLIGENT_TIERING',
ContentType=content_type)

    def base_path(self):
        return "https://" + self.bucket_name + ".s3-" + self.region + ".amazonaws.com/"
        pass

    def get_client(self):
        if self.client is not None:
            return self.client

        self.client = boto3.client('s3',
                                aws_access_key_id=self.key,
                                aws_secret_access_key=self.secret,
                                config=Config(
                                     max_pool_connections=50,
                                     region_name = self.region,
                                 ))

        return self.client
```

The rest of the providers have a very similar structure. There's nothing surprising here; the provider class takes the parameters and calls the client provided by the AWS Python library (called Boto, version 1.13.6 [47]) to perform simple operations. This simplicity makes it possible to implement many different types of storage providers. We don't rely on any special provider-specific feature here: just uploading files to an object storage with public HTTP access.

5.2.3.5 Video capture

Video capture is the part of the server that's responsible for the capturing of the video feed from the camera device, compressing, encoding and encrypting it and saving it to a file to be uploaded to the storage provider.

As mentioned in the *5.2.2 HTTP Live Streaming* section, the output of the encoding is a series of encrypted segment video files of approximately the same length and a playlist file that contains a list of them. This part heavily relies on FFMPEG (version 4.2.2 or later) [48], an extremely powerful video encoding library. The gist of the code below is simply about putting together an FFMPEG shell command that we execute as a subprocess.

The table below explains the parameters and flags used by the command.

`if is_raspberry_pi():` `  command = 'raspivid -n -w 640 -h 360 -fps 25 -vf -t 86400000 -b 1800000 -ih -o - | ffmpeg -y -i - '` `  codec = 'h264_omx'` `else:` `  command = 'ffmpeg -y -f avfoundation -framerate 30 -i "FaceTime:MacBook" '` `  codec = 'libx264'`	Here we define the input. For testing and development I used my Apple MacBook Pro that required a different definition of input than on the Raspberry Pi. Raspivid is a program bundled with Raspberry Pi OS that captures video input from a connected camera device. When called as seen here, it writes the captured video on the standard output, which is later picked up by FFMPEG. I opted for a 640x360 pixel resolution (`-w` and `-h` option), 25 frames per second, with a high bitrate (`-b` flag, it will be compressed later). Note that the process shuts down after 24 hours (`-t` flag) to avoid memory leaks or similar issues, but restarts immediately (see *5.2.3.8 Run* section). The platform also determines the compression library used. In both cases the H.264 compression standard [38] is used, which is widely understood by a variety of devices, including the target iOS and Android smartphones. However, `'h264_omx'` uses the GPU on Raspberry Pi to encode the video, sparing the CPU.
`command = command + \` `  ' -b 500k' + \`	This flag defines the desired bitrate for the output (compression level). 500 kilobits/s would result in approximately 200kB segment files given 3 second target segment time (see below). It's important that segment files are small enough so that they can be uploaded and downloaded faster than their average length in time, otherwise a queue of files to be uploaded would grow. Not counting latency, a 200kB file on a 20Mb/s home internet upload

	speed can be uploaded in about 0.08s. In my experimentation 500kb/s bitrate did not cause a considerable loss of quality for our purposes.
`' -vf scale=640:360' + \`	This flag uses the same resolution as the output from the Raspberry Pi camera's output.
`' -c:v ' + codec + \`	Selecting the compressions algorithms to be used on the video. See description above.
`' -pix_fmt yuv420p' + \`	This flag selects the YUV420p color encoding system. This is important because iOS only understands a particular set of color encodings.
`' -preset superfast' + \`	This flag selects an encoding preset that is relatively fast (but provides poorer compression performance) in order to make the video feed as real-time as possible.
`' -tune zerolatency' + \`	This flag fine-tunes the preset above for low latency streaming.
`' -vf "drawtext=fontfile=' + shlex.quote(FONT_PATH) + ': text=\'%{pts\:gmtime\:' + \`	The `vf` flag can be used in a variety of ways to apply video filters on the output. Here we use it to embed a timestamp caption on the video, indicating the current time. The first line calls the `drawtext` filter and defines the path to the font file, which is included in the code repository. The `gmtime` text formatter can be used to print a timestamp, relative to a starting time. The filter produces a text caption like this:

	
`str(start_timestamp) + '\:%d-%m-%Y %T}\':' + \`	Gmtime requires a start timestamp. We use the timestamp when the streaming starts, corrected by a timezone offset. This gives us text overlays on the video that indicate the current local time.
`' fontcolor=white: fontsize=h/10: x=(w-tw)/2: y=h-(2*lh):' + \`	Here we define the font color (white), the size relative to the height of the video feed and align it to the bottom center.
`' box=1: boxcolor=0x00000000@1: boxborderw=5: alpha=0.5"' + \`	We draw a black box around the caption and set its opacity to 50% to avoid covering up important details in the video.
`' -f hls' + \`	This flag tells FFMPEG that the output format should be HLS. See *5.2.2 HTTP Live Streaming* section.
`' -hls_time 3' + \`	This flag defines the target time for each segment in seconds. This means that a new segment file (and an updated playlist file) is written on the disk every 3 seconds or so while FFMPEG is running. Although intuitively this time should be as short as possible to achieve the lowest possible latency in the video feed between the server and the client, in practice setting it to lower than 3 seconds did not achieve shorter segments.
`' -hls_list_size 1' + \`	This flag defines that the generated playlist file should only contain a single item: the last segment. The playlist file is not uploaded to the storage so this could be any number.

`' -hls_start_number_source epoch ' + \`	This flag defines that FFMPEG should use the current Unix timestamp as the starting number of the first segment. This is important because it creates unique sement numbers and avoids overwriting existing segments as they are uploaded. Consecutive segment numbers will be incremented by one.
`' -hls_flags temp_file' + \`	This flag tells FFMPEG to write the current segment to a temporary file and atomically rename it once the segment is completed, instead of writing to the file as the video feed is coming in. This is important so that we avoid the uploader working with incomplete segment files.
`' -hls_segment_filename ' + shlex.quote(os.path.join(VIDEO_OUTPUT_PATH, "%012d.ts")) + \`	This flag defines the path where the segments should be saved as well as the format on the file name. The format is the segment number left-padded with zeros to make 12 characters, and the ".ts" extension indicating MPEGTS container format (see below). The padding is important because we compare the segment numbers as strings in the uploader (and not as numbers).
`' -hls_segment_type mpegts' + \`	We select MPEG transport stream as container format. This is common for HLS.
`' -hls_key_info_file ' + shlex.quote(SYMMETRIC_KEY_INFO_FILE_PATH) + \`	This flag defines that the segment files have to be encrypted with symmetric encryption, using the specified key info file. The key info file in turn has information about the key to be used for encryption. The key was uploaded by the smartphone client to the camera server during the configuration phase. This flag effectively completes one end of the end-to-end encryption of the video files. The encryption algorithm is AES-128 in Cipher block chaining (CBC) mode. The IV (initialization vector) is derived from the segment number and is contained in the segment file.

`    ' ' + shlex.quote(PLAYLIST_FILE_PATH)`	Finally, we tell FFMPEG to save the playlist file in a specified path.

While it seems easy to "just call a library" to achieve the video encoding and encryption, in practice configuring FFMPEG properly took a considerable amount of effort was perhaps the most time-consuming part of implementing the prototype. The long and complicated list of flags also indicate how powerful FFMPEG is.

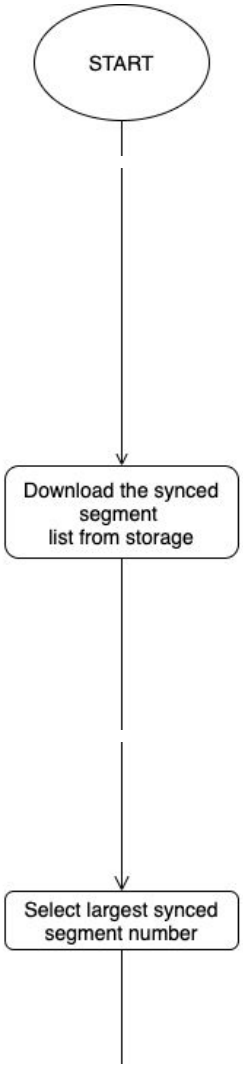
5.2.3.6 Video uploader

The purpose of the video uploader component is constantly monitoring the file system for new segment files output by the Video capture component (5.2.3.5) and uploading them to the cloud using one of the storage provider classes.

Albeit not strictly conceptually part of the video uploader, there's a secondary function being implemented here simply due to convenience; namely the extraction of thumbnails from the video segments. These thumbnails are an important part of the smartphone client user interface. Bundling this functionality with the uploader violates the single-responsibility principle (SRP) [49] and ought to be refactored, but I followed pragmatic considerations when I decided to do so.

The extraction of thumbnails raised a problem that I spent considerable time resolving both the server and the client part: while the video feed is end-to-end encrypted using HLS standards, the thumbnail files aren't. They are plain old image files (JPEG in our case) that needed to be encrypted similarly to the video segments, using the same key and encryption algorithm, and then decoded on the client before presenting.

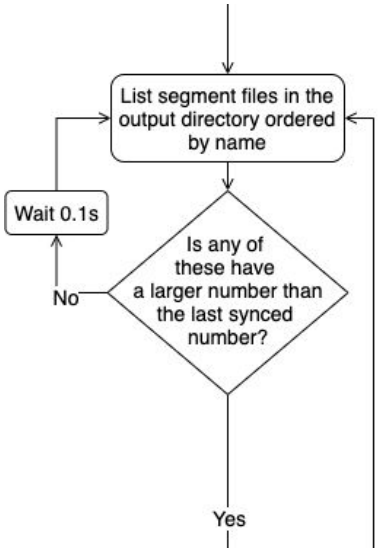
The flow chart below shows the main function of the uploader, with the most important pieces of code aligned to the flowchart nodes.



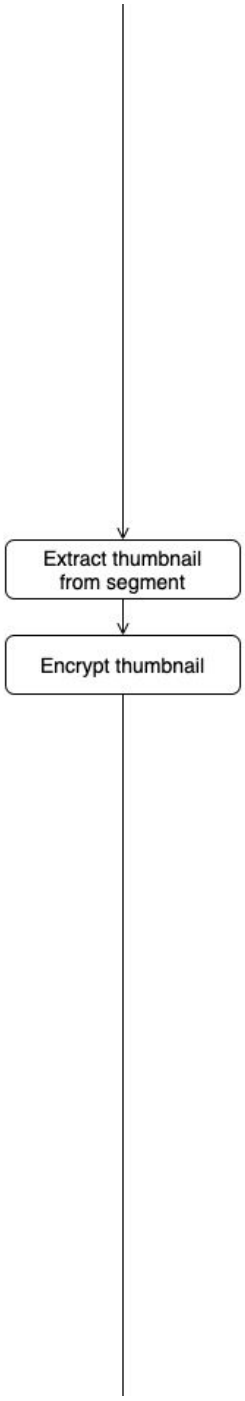
videouploader.py

```
def load_segment_list_from_storage(provider):
   base_path = provider.base_path()
   segment_list = []

   try:
       segment_list_string = urlopen(base_path +
'segment_list').read().decode('utf-8')
       segment_list = segment_list_string.split("\n")
   except HTTPError as http_error:
       if http_error.code not in [404, 400, 403]:
           raise http_error

   return segment_list
```

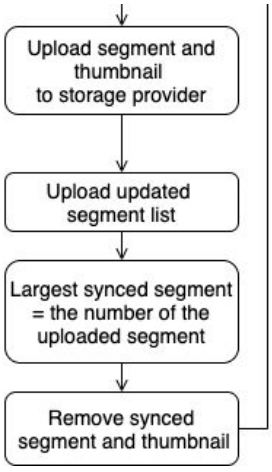
```
def last_segment_from_segment_list(segment_list):
   if len(segment_list) == 0:
       return 0
   return int(max(segment_list))
```




```
def get_next_segment_file_to_sync(last_synced_segment_number):
   while True:
       next_unsynced_file = None
       for filename in sorted(os.listdir(VIDEO_OUTPUT_PATH)):
           if filename.endswith(".ts") and int(filename.replace('.ts',
'')) > last_synced_segment_number:
               next_unsynced_file = os.path.join(VIDEO_OUTPUT_PATH,
filename)
               break

       if next_unsynced_file is None or not
os.path.isfile(PLAYLIST_FILE_PATH):
           time.sleep(0.1)
           continue

       return next_unsynced_file
```




```
def extract_thumbnail(video_file):
    segment_name = os.path.basename(video_file).replace('.ts', '')
    thumbnail_filename = os.path.join(VIDEO_OUTPUT_PATH, segment_name +
'.jpg')
    unencrypted_thumbnail_filename = os.path.join(VIDEO_OUTPUT_PATH,
segment_name + '.unencrypted.jpg')
    playlist_filename = os.path.join(VIDEO_OUTPUT_PATH, segment_name +
'.m3u8')

    playlist = """\
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:3
#EXT-X-MEDIA-SEQUENCE:1
#EXT-X-KEY:METHOD=AES-128,URI="{key_file}"
#EXTINF:3,
{segment_file}
#EXT-X-ENDLIST
    """.format(key_file=SYMMETRIC_KEY_FILE_PATH, segment_file=video_file)

    with open(playlist_filename, 'w') as playlist_file:
        playlist_file.write(playlist)

    if not os.path.isfile(unencrypted_thumbnail_filename):
        cmd = 'ffmpeg' + \
              ' -allowed_extensions ALL' + \
              ' -i ' + shlex.quote(playlist_filename) + \
              ' -vframes 1 -q:v 2 ' + unencrypted_thumbnail_filename

        process = subprocess.Popen(cmd, shell=True,
stdout=subprocess.PIPE)
        process.wait()

    if not os.path.isfile(thumbnail_filename):
        encrypt_file(SYMMETRIC_KEY_FILE_PATH,
unencrypted_thumbnail_filename, thumbnail_filename)
    os.remove(unencrypted_thumbnail_filename)
    os.remove(playlist_filename)
    return thumbnail_filename
```




```
provider.sync_file(key=os.path.basename(segment_filename),
                   path=segment_filename,
                   content_type='video/MP2T')

provider.sync_file(key=os.path.basename(thumbnail_filename),
                   path=thumbnail_filename,
                   content_type='image/jpeg')

segment_name = os.path.basename(segment_filename).replace('.ts', '')
segment_list.append(segment_name)
last_synced_segment_number = last_segment_from_segment_list(segment_list)

provider.sync_string(key='segment_list',
                     content="\n".join(segment_list),
                     content_type='text/plain')

os.remove(segment_filename)
os.remove(thumbnail_filename)
```

A few notable points about the code above:

- The segment list is a simple text file with the names of each consecutive segment in each line, uploaded to the provider unencoded - it contains no sensitive information. It's used both by the camera server and the smartphone client to keep track of the segments that are stored on the provider.
- Once again we use FFMPEG to extract the thumbnail. Note that we are passing a playlist file and not the segment file as input, because the segment file is encrypted at this point and information from the playlist file is needed to decode it. The playlist file is created on the fly, containing just the one segment and the path to the key information file
- The playlist file output by FFMPEG is never uploaded to the provider. It is re-created on the fly in the smartphone client. See *5.2.3.12 Video view*.

5.2.3.7 Encryption

As described in the *5.2.2 HTTP Live Streaming* and *5.2.3.5 Video capture* section, the video segments are encoded using AES-128 symmetric encryption, using the key uploaded by the smartphone client to the camera server during the configuration phase. However, the thumbnail files are unencrypted by default, so we have to encrypt them using the same key and encryption algorithm.

crypto.py

```
import binascii
import os, random, struct
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

def encrypt_file(key_filename, in_filename, out_filename):
    chunksize = 64 * 1024

    with open(key_filename, 'rb') as key_file:
        key = key_file.read(16)

    iv = get_random_bytes(16)

    encryptor = AES.new(key, AES.MODE_CBC, iv)

    with open(in_filename, 'rb') as infile:
        with open(out_filename, 'wb') as outfile:
            outfile.write(binascii.hexlify(iv))
            outfile.write( b"\0\0\0\0")

            while True:
                chunk = infile.read(chunksize)
                if len(chunk) == 0:
                    break
                elif len(chunk) % 16 != 0:
                    chunk += b"\0" * (16 - len(chunk) % 16)

                outfile.write(encryptor.encrypt(chunk))
```

The code uses AES-128 [39] with Cipher block chaining (CBC) and 64 kilobyte block size. Once again we use pycrypto (2.6.1) [41] to do the heavy lifting. There are two notable things here.

The first is the random initialization vector (IV) prepended to the file. It might look like an odd choice to use the hexadecimal representation (`binascii.hexlify`) to store it in an otherwise binary file. The careful reader might also pick up on the strange 4 zero-bytes following the IV. The answer, similarly to many other oddities in computing, is JavaScript.

The smartphone client's framework, React Native is running on JavaScript. Unfortunately JavaScript has no proper binary string type: its strings are UTF-16 encoded texts [30]. This means that they are simply not suitable to store an arbitrary binary string, since some random binary characters might end up causing illegal code errors when interpreted as UTF-16. Hence,

all binary data needs to be represented in JavaScript in a different base than 256. Hexadecimal (base 16) or Base-64 are common choices.

The AES encryption library used by the client therefore expects keys and IVs to be passed as hexadecimal strings, representing 4 bits with each character. However, the library used to read files from the disk (e.g. the downloaded image file) uses Base-64, representing 6 bits in each character. Because of this, the length of the IV suffix needs to be divisible by 6 bits. This is why 4 extra bytes are added to the 32 original bytes (16 bytes encoded in hexadecimal), making it 36 bytes, or 288 bits, divisible by 6 bits. The quirks of working with languages with missing types.

Note also that the last encoded blocks are padded with zero-bytes to give 64 kilobytes total. Since we don't store the data length anywhere, this will be part of the decoded ciphertext, but because we are solely using this code to encrypt JPEG files this is not an issue; these padding bytes will be ignored by the image library on the client.

5.2.3.8 Run

The code that glues everything together can be found in `run.py`. It uses process forking to run different components of the server (see Figure 17), and restarts subprocesses if any issue occurs.

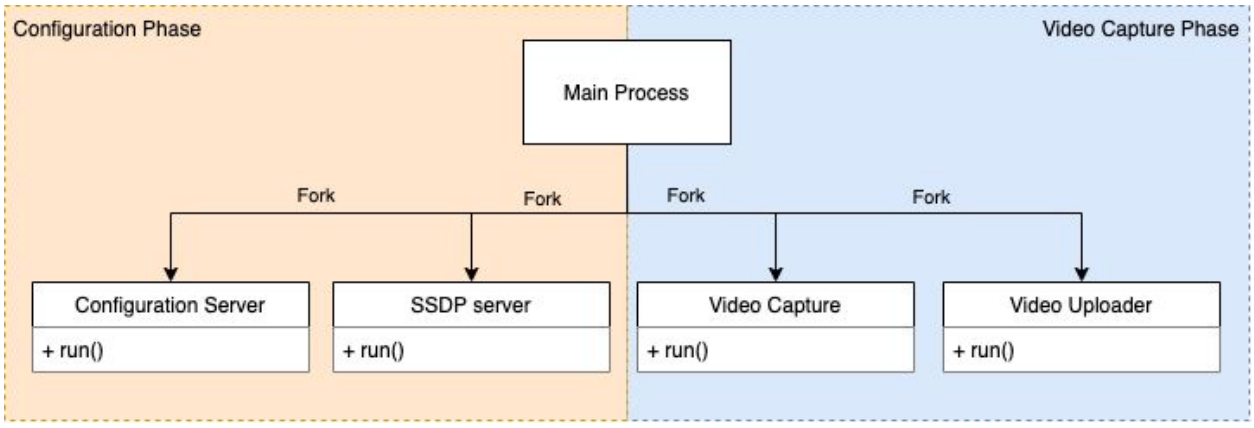


Figure 17: Subprocess tree

The code is not very exciting but can be found in the Appendix (*9.4.1 Camera Server*). One notable thing is the transition from the Configuration Phase to the Video Capture Phase. When the Configuration Server process exists with code 0 (success), the main process knows that the configuration is successful and it can transition into the Video Capture mode. It shuts down the SSDP process and starts running the Video Capture and Video Uploaded processes.

The default phase is the Configuration phase, unless the Main process detects that the server has already been configured from the smartphone client.

If the reader is not yet sick of reading source code, it's time to transition to the client code. The order by which we will examine the components will match the order of their server counterparts above as much as possible. Before we dive into the client code, it is recommended to review the chart in the *4.4.2.4 Storyboard* section, since most of the code in the client is organized around screens.

5.2.3.9 QR code reader

Let's begin with the QR code reader. The objective is to read the public key of the camera server from the code printed on its case.

Figure 18: QR code scanner screen, with the viewfinder pointed at the QR code mounted on the camera's case

Before we go too deep into the woods of React Native, it's worth mentioning that it relies heavily on two unique concepts:

- Components. These are self-contained, reusable pieces of code with UI focus, rendering parts of the user interface. They have a life cycle, properties, and state. They tend to be nested in each other, using JSX notation, which is, simply put, JavaScript mixed with HTML-like syntax.
- Unidirectional data flow. he view (user interface) of the component is determined by its state. The view can trigger actions. Actions might manipulate the state of the component,

which, in turn, updates the view and subcomponents. The system relies on one-way bindings which makes it less error-prone. See Figure 19

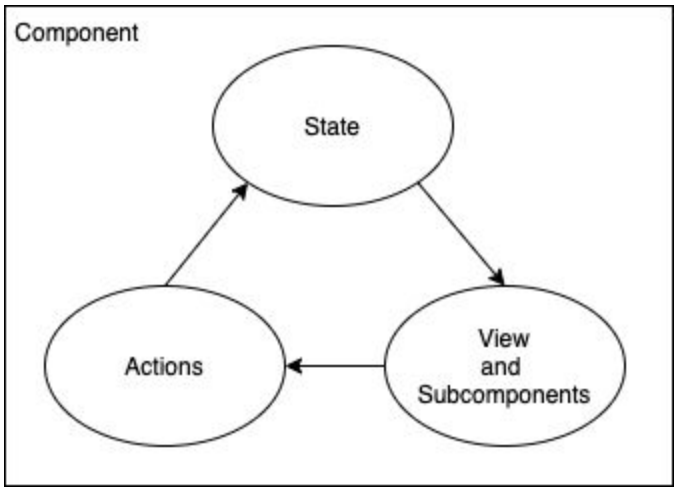


Figure 19: React native unidirectional data flow

Let's look at how it works in the QR code reader screen.

screens/CodeScanner.js (partial)

```
render() {

   if (this.state.hasPermission === null) {
       return <Text>Requesting for camera permission</Text>;
   }
   if (this.state.hasPermission === false) {
       return <Text>No access to camera</Text>;
   }


   return (
       <View
           style={{
               flex: 1,
               flexDirection: 'column',
               justifyContent: 'flex-end',
           }}>
           <BarCodeScanner
               onBarCodeScanned={this.state.scanned ? undefined :
this.handleBarCodeScanned.bind(this)}
```

```
                style={StyleSheet.absoluteFillObject}
                barCodeTypes={[BarCodeScanner.Constants.BarCodeType.qr]}
            />
        </View>
    );
}
```

If the component does not have the necessary permissions, the appropriate text is shown. Otherwise, it renders the `BarCodeScanner` component. This is provided by the expo-barcode-scanner (8.1.0) [50] library. It's basically a camera viewfinder with barcode reading capabilities.

Once the component scans a barcode, it calls the handleBarCodeScanned method.

screens/CodeScanner.js (partial)

```
handleBarCodeScanned ({ type, data }) {
    function validatePublicKeyBase64 (data) {
        var base64regex = /^([0-9a-zA-Z+/]{4})*(([0-9a-zA-Z+/]{2}==)|([0-9a-zA-Z+/]{3}=))?$/;

        if (!base64regex.test(data)) {
            return false;
        }
        if (data.length !== 392) {
            return false;
        }
        return true;
    }

    if (!validatePublicKeyBase64(data)) {
        console.log('Invalid QR code: ' + data);
        return;
    }

    this.setState({ scanned: true });

    this.props.navigation.navigate('DeviceSearch', {
        publicKey: data,
    });
}
```

The data from the barcode is validated for its character set (valid Base-64) and length (according to the key size and DER specifications [44]). If the code is valid and looks like a public key, the code navigates to the next screen, the Device Search, passing the public key.

5.2.3.10 Device Search

Figure 20: Device search screen

This screen implements the SSDP client that tries to find the camera server on the same network based on the service type and its private key. For details on the SSDP protocol see the *5.2.3.2 SSDP server* section.

The component's user interface isn't very interesting, the interesting part boils down to the `searchDevice` method.

screens/DeviceSearch.js (partial)

```
this.socket = dgram.createSocket('udp4');

// ... omitted code

var message = "M-SEARCH * HTTP/1.1\r\n" +
   "HOST: 239.255.255.250:1901\r\n" +
   "MAN: \"ssdp:discover\"\r\n" +
   "MX: 1\r\n" +
   "ST: ssdp:choicecam";

this.socket.send(base64.encode(message), 0, message.length, 1901, '239.255.255.250',
function(err) {
   if (err) throw err;
});
```

While there are a number of SSDP libraries available for React Native, I found these buggy or poor quality. Since I only needed to implement a small subset of functionalities, I decided to do this from scratch. As the code above shows, the request is constructed on the fly as a simple string according to the SSDP specifications. We send a search request for the `ssdp:choicecam` service type to the multicast IPv4 address, through UDP. Note that for UDP requests I use the react-native-udp (3.1.0) [51] library.

screens/DeviceSearch.js (partial)

```
this.socket.on('message', function(messageBytes, serverInfo) {
   const message = byteArrayToString(messageBytes);
   const headers = parseHeader(message);
   if (!headers || !headers['usn']) {
       return;
   }

   if (headers['usn'] !== publicKeyToSearch) {
       return;
   }

   this.stopSearch();
   this.props.navigation.navigate('DeviceSetup', {
       publicKey: publicKeyToSearch,
       ipAddress: serverInfo.address,
   });

}.bind(this));
```

Next we attach an event listener to the UDP socket, responding to incoming messages. Once we receive a message, we parse it. If it's not a `NOTIFY` method, we discard it (in `parseHeader` method). Otherwise we return the parsed headers as an object. Next, we compare that the USN (Unique Service Name) is the same as the public key read from the QR code. If it matches, we go to the next screen (Device Setup), passing the public key and the IP address of the camera that we now know.

Note that this screen tends to find the camera server on the network in a split second, so it's barely visible on the demonstration video.

5.2.3.11 Device Setup

Once again, the abstraction of the infrastructure provider (storage in this case) is the pinnacle of this project. There may be many security camera apps, but you don't see a screen like in Figure 21 in any of them. In fact, you don't see a similar screen in any app that requires storing personal data in the cloud - or at least not to my knowledge.

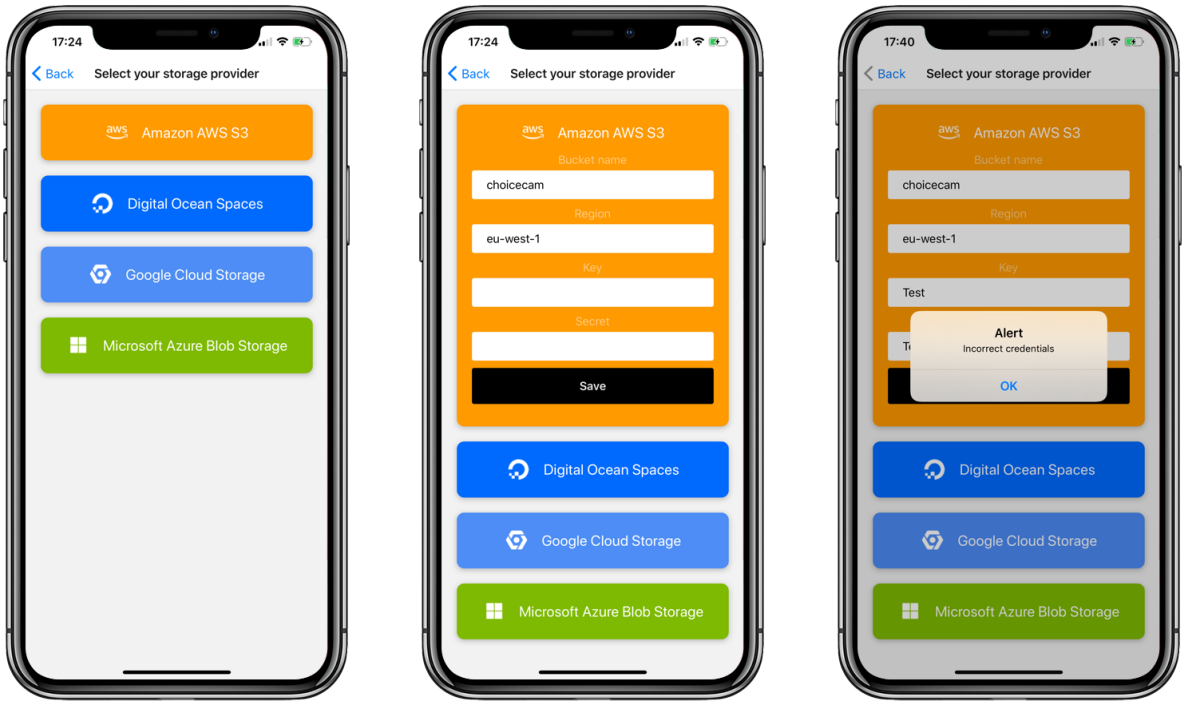


Figure 21: Different states of the Device Setup screen that lets the user select where to store their personal data.

I selected the following 4 providers based on the criteria that they have a compatible product (object storage with public web read option) and availability of Python client libraries.

- Amazon Web Services (AWS) Simple Storage Service (S3) [52]
- Google Cloud Storage [53]
- Microsoft Azure Blob Storage [54]
- Digital Ocean Spaces [55]

Different providers require a different set of credentials to work. For example, AWS needs an access key, a secret key, a region and a bucket name while Azure only needs a container name and a connection string. We handle this with inheritance and polymorphism in the code:

components/provider-forms/ProviderForm.js (partial)

```
class ProviderForm extends React.Component {

   // Omitted code...

   render() {
       const title = this.getTitle();
       const fields = this.getFields();
       const backgroundColor = this.getBackgroundColor();
       const logo = this.getLogo();

       // Omitted code, rendering the component based on the methods of the concrete
provider
    }
}
```

`ProderForm` is a (conceptually) abstract class. The methods `getTitle`, `getFields`, etc are implemented in the concrete class and called here using dynamic dispatch.

components/provider-forms/AwsProviderForm.js

```
class AwsProviderForm extends ProviderForm {
   constructor(props) {
       super(props);
   }

   getType() {
       return 'aws';
   }
```

```
    getTitle() {
        return 'Amazon AWS S3';
    }

    getFields() {
        return [
            {'name': 'bucket_name', 'label': 'Bucket name', 'default': 'choicecam'},
            {'name': 'region', 'label': 'Region', 'default': 'eu-west-1'},
            {'name': 'key', 'label': 'Key', 'default': 'choicecam'},
            {'name': 'secret', 'label': 'Secret', 'default': 'choicecam'},
        ]
    }

    getBackgroundColor() {
        return '#ff9a00';
    }

    getLogo() {
        return require('../../images/aws.png');
    }
}
```

It's very easy to define new providers; one simply needs to extend the `ProviderForm` component and implement these methods.

Once a provider is selected and the form is filled with the credentials, they are sent to the camera server along with a randomly generated key for the symmetric encryption of the video and thumbnail files. This request is encrypted with the public key of the camera.

```
Aes.randomKey(16).then(function (symmetricKey) {
    var message = {
        'symmetric_key': symmetricKey,
        'provider_data': providerData
    };

    message = JSON.stringify(message);
    publicKey = "-----BEGIN PUBLIC KEY-----" + publicKey + "-----END PUBLIC KEY-----";

    RSA.encrypt(message, publicKey)
        .then(encodedMessage => {
            return fetch('http://' + ipAddress + ':8000/', {
                method: 'POST',
                body: encodedMessage
```

```
            });
        }).catch(function () {
            throw new Error("Unexpected error happened. Please try again.");
        })
        .then(function (response) {
            return verifyResponse(response, publicKey)
        }).catch(function () {
            throw new Error("The identity of the webServer could not be verified.");
        }).then(function (result) {
            var response = result.response;
            var responseText = result.responseText;
            if (!response.ok) {
                throw new Error(responseText);
            }

            saveBasePathAndKey.call(component, responseText, symmetricKey);
        }).catch(function(error) {
            showError.call(component, error.message);
        });
});
```

The code above using several chained then and catch callbacks is often referred to as "promise hell", which results from the asynchronous nature of Javascript.

For AES encryption related functionalities (e.g. generating key) I use a library called react-native-aes-crypto [56]. Note that this library turned out to be buggy and not being able to handle binary strings, so I had to fork and fix it [57].

For RSA related methods (encryption, verifying signature) I use a library called react-native-rsa-native (2.0.0) [58].

The received response from the Configuration Server is then verified using the public key of the camera server (it was signed using the private key). If it's found to be authentic, the base path returned from the server as well as the generated symmetric key are saved on the device to an encrypted storage partition using the library expo-secure-store (8.1.0) [59]. On iOS this secure store can be backed up to the user's iCloud storage, so in case of the loss or theft of the device the configuration is automatically recovered on a new device.

At this point the client and the server are both configured, we can move on to the video view.

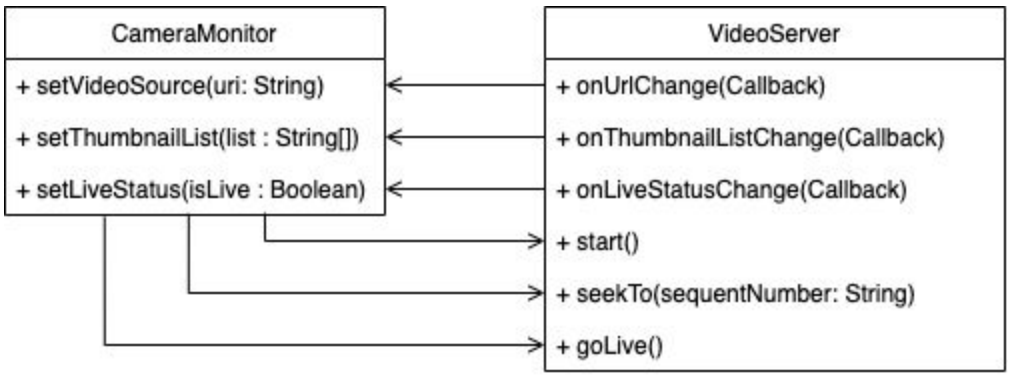
5.2.3.12 Video View

This is the default screen of the smartphone application after the configuration has been completed. Its functionality is to show the live video feed from the camera, and allow the user to replay recorded video from an arbitrary timestamp (see Figure 22).

Figure 22: The Video View screen in its two states: live or playback

The screen consists of two parts: the video view on the top and the list of past thumbnails on the bottom in an infinitely scrollable list. Tapping on any thumbnail takes the user back to that timestamp in the recording.

The code of the screen is relatively simple, but it relies heavily on the `VideoServer` class through a series of callbacks and method calls. The `VideoServer` does most of the heavy lifting of constructing HLS playlists.



The `VideoServer` class does the following:

1. Starts a web server only accessible from localhost. This is needed because iOS cannot load HSL files from a local URI. This way we "pretend" that it's a remote URI.
2. Downloads the segment list from the storage provider (see *5.2.3.6 Video uploader* section) and keeps it updated by polling it every second.
3. When the segment list changes, it creates a M3U8 playlist file containing the new segments (see *5.2.2 HTTP Live Streaming* section)
4. Notifies the `CameraMonitor` component to update the URI of the video player from the local web server, as well as the list of thumbnails.

Let's see these parts one by one, with some remarks.

`VideoServer.js (partial)`

```
startWebServer() {
   var documentRoot = FileSystem.documentDirectory.replace('file://', '');
   this.webServer = new StaticServer(this.PORT, documentRoot, {localOnly : true});
   this.webServer.start();
}
```

This part relies on react-native-static-server (0.4.2) [60] and expo-file-system (8.1.0) [61] and poses a small security risk, because, although the web server is only available locally (`localOnly` flag), a malicious app on the same device might be able to access the files it host, including the symmetric key, given that it can guess the random URI of the playlist file. See *5.3 Security* section for more details.

Next, we download the segment list.

VideoServer.js (partial)

```
refreshSegmentList()
{
   return new Promise(function (resolve, reject) {
       fetch(this.basePath + 'segment_list?cachebust=' + Math.random())
           .then(response => response.text())
           .then((response) => {
               var segmentList = response;
               if (segmentList === "") {
                   return;
               }

               var segmentNames = segmentList.split("\n");
               var segments = [];
               for (var i = 0; i < segmentNames.length; i++) {
                   var segmentName = segmentNames[i];
                   var item = {
                       'videoFile': this.basePath + segmentName + '.ts',
                       'thumbnailFile': this.basePath + segmentName + '.jpg',
                       'sequence': i + 1,
                   };

                   segments.push(item);
               }

               this.segments = segments;
               resolve();
           })
           .catch(reject);
   }.bind(this));
}
```

As seen above, the request to get the segment list is a plain old HTTP request to the storage provider, using the base path obtained from the camera server during the configuration phase. The segment list contains single lines of segment numbers padded with zeros, see *5.2.3.6 Video uploader* section. The segment data that is constructed from the segment list will look something like this:

```
[
    {
        "videoFile":
"https://choicecam.s3-eu-west-1.amazonaws.com/001590609842.ts",
```

```
        "thumbnailFile":
"https://choicecam.s3-eu-west-1.amazonaws.com/001590609842.jpg",
        "sequence": 1
     },
     {
        "videoFile":
"https://choicecam.s3-eu-west-1.amazonaws.com/001590609843.ts",
        "thumbnailFile":
"https://choicecam.s3-eu-west-1.amazonaws.com/001590609843.jpg",
        "sequence": 2
     },
     …
]
```

Once we have the segment list, we can construct the HLS playlist file.

VideoServer.js (partial)

```
rewritePlaylistFile() {
   const keyInBase64 = hexToBase64(this.key);
   var playlistStartSequence;
   if (null !== this.currentSequence) {
       playlistStartSequence = this.currentSequence;
   } else {
       var tailSizeForLiveVideo = 4;
       playlistStartSequence = Math.max(1, this.segments.length - tailSizeForLiveVideo + 1);
   }

   var playlistFile = "#EXTM3U\n" +
       (null !== this.currentSequence ? "#EXT-X-PLAYLIST-TYPE:EVENT\n" : "") +
       "#EXT-X-VERSION:3\n" +
       "#EXT-X-TARGETDURATION:8\n" +
       "#EXT-X-ALLOW-CACHE:NO\n" +
       "#EXT-X-MEDIA-SEQUENCE:" + playlistStartSequence + "\n" +
       "#EXT-X-KEY:METHOD=AES-128,URI=\"data:text/plain;base64," + keyInBase64 + "\"\n";

   for (var i = 0; i < this.segments.length; i++) {
       var segment = this.segments[i];

       if (segment.sequence < playlistStartSequence) {
           continue;
       }
       playlistFile = playlistFile + "#EXTINF:8.333333,\n" +
```

```
            this.segments[i].videoFile + "?cachebust=" + this.playlistFilename + "\n";
    }

    var uri = FileSystem.documentDirectory + this.playlistFilename;
    return FileSystem.writeAsStringAsync(uri, playlistFile);
}
```

Before we can write the playlist file, we need to make sure the encryption key is somehow accessible to the video client, as per HLS specifications [36]. The `#EXT-X-KEY` header requires a URI where the key can be downloaded from. Hosting the key in from the local static server is an option, but it increases the security risk associated with the server itself, see *5.3 Security*. Instead, I decided to create a data-URI that contains the data itself. Data URIs support plain text and Base-64 encoding. Since the key is a random binary string, Base-64 was the only choice. The key in the client is stored in hexadecimal notation, because Javascript does not support binary strings, only UTF-16 encoded strings [30]. Converting hexadecimal (Base-16) to Base-64 without the use of binary string was a fun coding challenge, the code is made available in the Appendix, under `Utilities.js` in 9.4.2.

The rest of the code constructs the M3U8 files according to the HLS specifications (see *5.2.2 HTTP Live Streaming*). When the video view is in "live" mode, we have a moving window of the 4 last segments in the playlist, constantly updating as new segments become available. In case of a playback, we list all segments starting with the one selected by the user (`this.currentSequence`), and use the `EVENT` playlist type header.

This covers the main functionality of the `VideoServer` class.

5.2.3.13 Encrypted thumbnails

One uncovered area in the code is the one that deals with thumbnails. Thumbnail URLs are constructed from the segment number from the segment list and the base URL, for example "001590609842" becomes "https://choicecam.s3-eu-west-1.amazonaws.com/001590609843.jpg". However, we should remember that this is an encrypted file. For details on the encryption see the *5.2.3.6 Video uploader* section.

React Native lends itself to creating an encrypted image component that works exactly like a "normal" image component, in that it takes a URL and displays an arbitrary image, except the new component takes the URL of an encrypted image file.

Consider the JSX code that renders the thumbnail list user interface:

screens/CameraMonitor.js (partial)

```
<FlatList
   style={styles.list}
   data={this.state.thumbnailList}
   renderItem={({item, index}) =>
       <TouchableOpacity onPress={ () => this.onListItemPress(item)}>
           <View style={[styles.listItem, index === 0 ? {display: 'none'} : {}]}>
               <Aes128EncryptedImage
                   style={styles.thumbnail}
                   source={{uri: item.thumbnailFile}}
                   symmetricKey={this.state.key}/>
           </View>
       </TouchableOpacity>
   }
   keyExtractor={(item, index) => item.sequence}
/>
```

It's a list component, where each individual list item is an `Aes128EncryptedImage` (wrapped in touch handlers). I implemented this component for this project, handling the encryption format described above. It behaves exactly like an `Image` component.

The most interesting part of the `Aes128EncryptedImage` component is where the image is downloaded and decrypted:

components/Aes128EncryptedImage.js (partial)

```
FileSystem.downloadAsync(
   uri,
   localUri
)
   .then(function (result) {
       return FileSystem.readAsStringAsync(result.uri, {
           encoding: FileSystem.EncodingType.Base64
       });
   }.bind(this))
   .then(function (contents) {
       if (!this.state.key) {
           throw new Error("Missing symmetric key.");
       }
       var iv = contents.substring(0, 48); // First 36 bytes in base64, chopped to 32
       iv = base64.decode(iv).substring(0, 32);
```

```
        var cipherText = contents.substring(48);
        return Aes.decrypt(cipherText, this.state.key, iv);
    }.bind(this))
    .then(function (decryptData) {
        return FileSystem.writeAsStringAsync(localUriDecoded, decryptData, {
            encoding: FileSystem.EncodingType.Base64
        });
    }.bind(this))
    .then(function (contents) {
        this.setState({decryptedImagePath: localUriDecoded});
    }.bind(this))
```

This is the flipside of the Base-64/Base-16 initialization vector issue I described in the *5.2.3.6 Video uploader* section.

This concludes the review of the most important parts of the code of the prototype. The full code is available in the Appendix section 9.4 and as an attachment in the auxiliary files.

5.3 Security

Due to the nature of the product - home security device with a life feed to the user's home - security is a concern in the prototype. While delivering an airtight product is not the main focus of this project (demonstrating the separation of storage infrastructure from the service is), here I list several attack vectors to consider.

5.3.1 Stealing the public key of the camera

It is certainly possible to steal the public key of the camera, e.g. by someone in the factory, since it's glued on the camera case. While public keys are meant to be public (*nomen est omen*), this would theoretically allow someone to configure the camera before the user gets to configure it, gaining access to the video feed as soon as the user connects the camera.

This would, however, be easily detected because the user could not configure the camera.

Note that precisely because of this risk, the camera configuration services (SSDP server, configuration server) shut down after the camera is configured and they can only be restored with factory reset, so that this attack is not possible once the user has set up the camera.

The risk can be further mitigated by tamper-proofing the packaging (considering a real, marketable product)

5.3.2 Man-in-the-middle while configuring the camera

While difficult to carry out, one could theoretically replace the sticker with the public key on the camera before it's delivered to the user and run a "decoy" service using the new public key, mimicking the camera. The decoy service would act as an intermediary between the client and the camera, and steal the symmetric key or the provider credentials.

Since the attacker would need to be connected to the same home network as the user, this is relatively low risk. It can be further mitigated by tamperproofing the package, or two-way authentication (e.g. the camera would also need to read the public key of the client, generated on the fly by the application).

5.3.3 Accessing the local webserver on the client

As described in *5.2.3.12 Video View*, the smartphone application has to run a local web server to serve the HLS playlist file (only serving to localhost). While other applications cannot read files stored by another application on the same device, theoretically, a malicious app could

access the web server locally and load the playlist file, gaining access to the symmetric key as well. The playlist file uses a random file name so this is somewhat difficult, but not impossible. To mitigate this risk, one could use a much larger key space for the name (e.g. UUID) and properly clean up playlist files, which is currently not done.

5.3.4 Accessing the camera device

Finally, if someone has access to the camera after it's been configured, they can easily steal the symmetric key and provider credentials that are stored on its hard drive. While certainly a risk, if someone has physical access to the camera, the user has a bigger problem in terms of security.

It's worth noting that the camera prototype has achieved end-to-end encryption for the video files, meaning that even the storage provider cannot access sensitive files.

5.4 Costs of storage

One aspect of the new architecture (separating the service from the infrastructure) is that it changes the total cost structure of the product. Some online services (Facebook, Gmail, etc) are currently free (or better put: subsidized by the user's personal information), but this would certainly change if the user had to pay for the infrastructure of storing their personal data. This is a trade-off between the improved privacy and the price of the service, which some users might or might not want to make (my survey did not cover this, regrettably). However, many people are already paying for some sort of storage service, like Dropbox or iCloud to store their backup files, photos, etc. Some of these services might be viable as Choice Cam storage providers.

As per our benchmark product, NestCam, the storage is not free. The NestCam Indoor hardware currently retails for approximately 140 € in Spain [62]. However, this does not include the storage service, the user only gets live video feed without playback in the base version. The playback service costs 5 € per month for a 30 day playback window, or 10 € for a 60 day playback window [63]. Note that the lookback window only stores "events", that is, footage with movement or sound.

How does that compare to our solution?

5.4.1. Storage needs

We chose to compress the video footage to 500 kilobits/s bitrate (see *5.2.3.5 Video capture*), or 62.5 kilobytes per second. Accounting for thumbnail images and the failure to always hit the bitrate, we can safely calculate with 100 kB per second, or 360 MB per hour. Considering a 30 day continuous video playback window this gives 259,200 MB or ~260 GB storage requirement. For this calculation we ignore the size of the segment list file.

Typically cloud storage providers charge for the size of the data stored as well as the bandwidth and operations required to upload/download it. Due to the nature of the product I will assume minimal download cost. I assume the size of the uploaded data per month is the same as the size of the stored data and count a single operation for upload every 3 seconds, or 864,000 operations per 30 days.

5.4.2 Comparison

Below is a comparison of the different storage providers. I calculated with the default storage options without special availability tiering and assumed European regions where applicable. I used USD as currency because it was the only currency available everywhere.

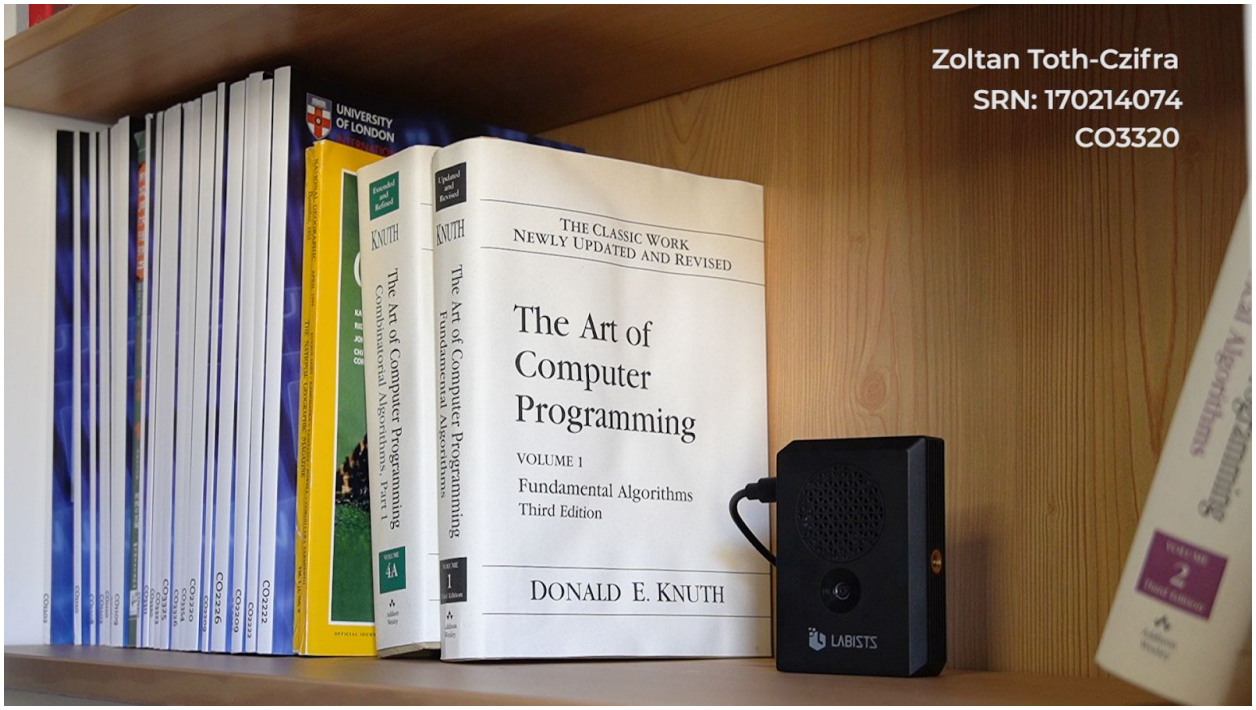
	Storage	**Upload**	**Total per 30 days**
Amazon AWS S3 [64]	5.98 USD	4.32 USD	10.30 USD
Google Cloud Storage [65]	5.20 USD	4.30 USD	9.50 USD
Microsoft Azure Blob Storage [66]	5.10 USD	4.67 USD	9.77 USD
Digital Ocean Spaces [67]	5.10 USD	0	5.10 USD

Digital Ocean turns out to have the best value, beating major infrastructure-as-a-service providers. It also beats the benchmark NestCam's offering (5 EUR = 5.67 USD at the time of writing this). Since Digital Ocean is the only provider on the list that does not own an advertisement business, it is perhaps the best choice in terms of privacy as well.

Note that a home security camera by nature requires the storage of a large amount of data. Other cloud services might have a much smaller footprint.

5.5 Demonstration

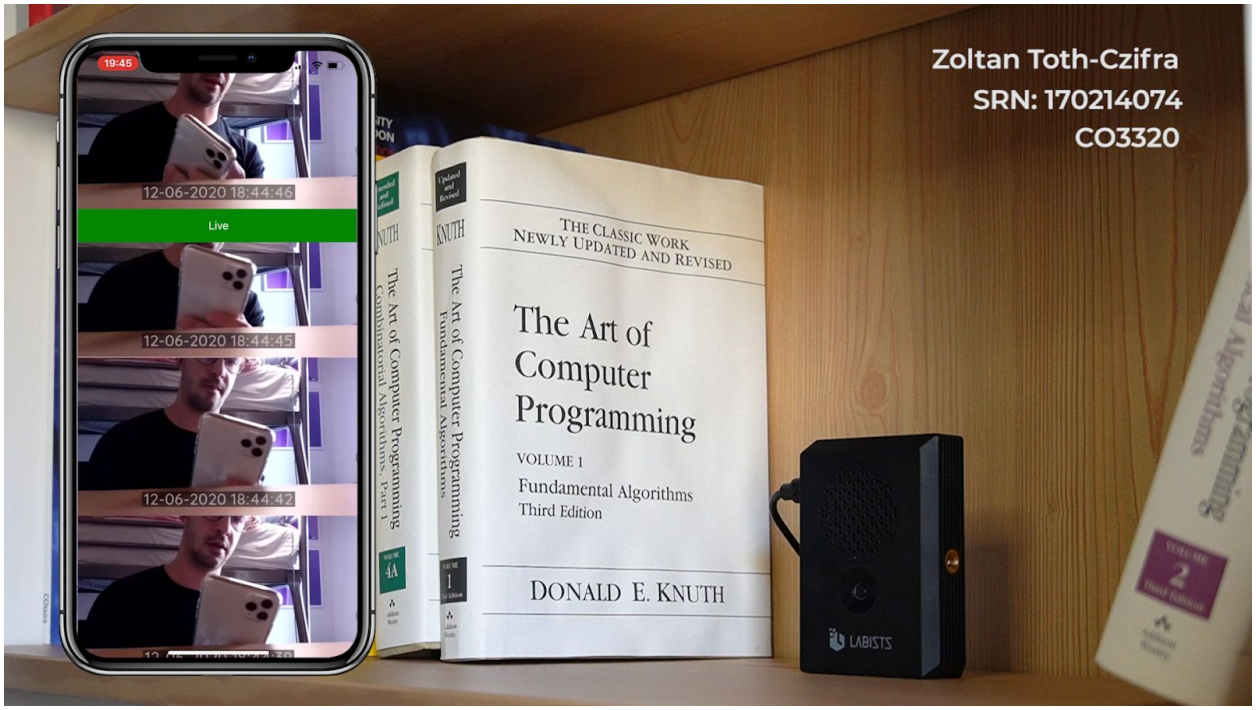
To the auxiliary files of this project I attach a video demonstrating the prototype. For the record I leave here some key frames from the video.




Zoltan Toth-Czifra
SRN: 170214074
CO3320


Zoltan Toth-Czifra
SRN: 170214074
CO3320
UNIVERSITY OF LONDON
VOLUME 1
Third Edition
DONALD E.


19:42
Back
Select your storage provider
Amazon AWS S3
Digital Ocean Spaces
Google Cloud Storage
Microsoft Azure Blob Storage
Zoltan Toth-Czifra
SRN: 170214074
CO3320


19:45
12-06-2020 18:44:46
Live
12-06-2020 18:44:45
12-06-2020 18:44:42
THE CLASSIC WORK
NEWLY UPDATED AND REVISED
The Art of Computer Programming
VOLUME 1
Fundamental Algorithms
Third Edition
DONALD E. KNUTH
Zoltan Toth-Czifra
SRN: 170214074
CO3320

Additionally, I show on Figure 23 that I have full visibility and control over the data stored by Choice Cam.

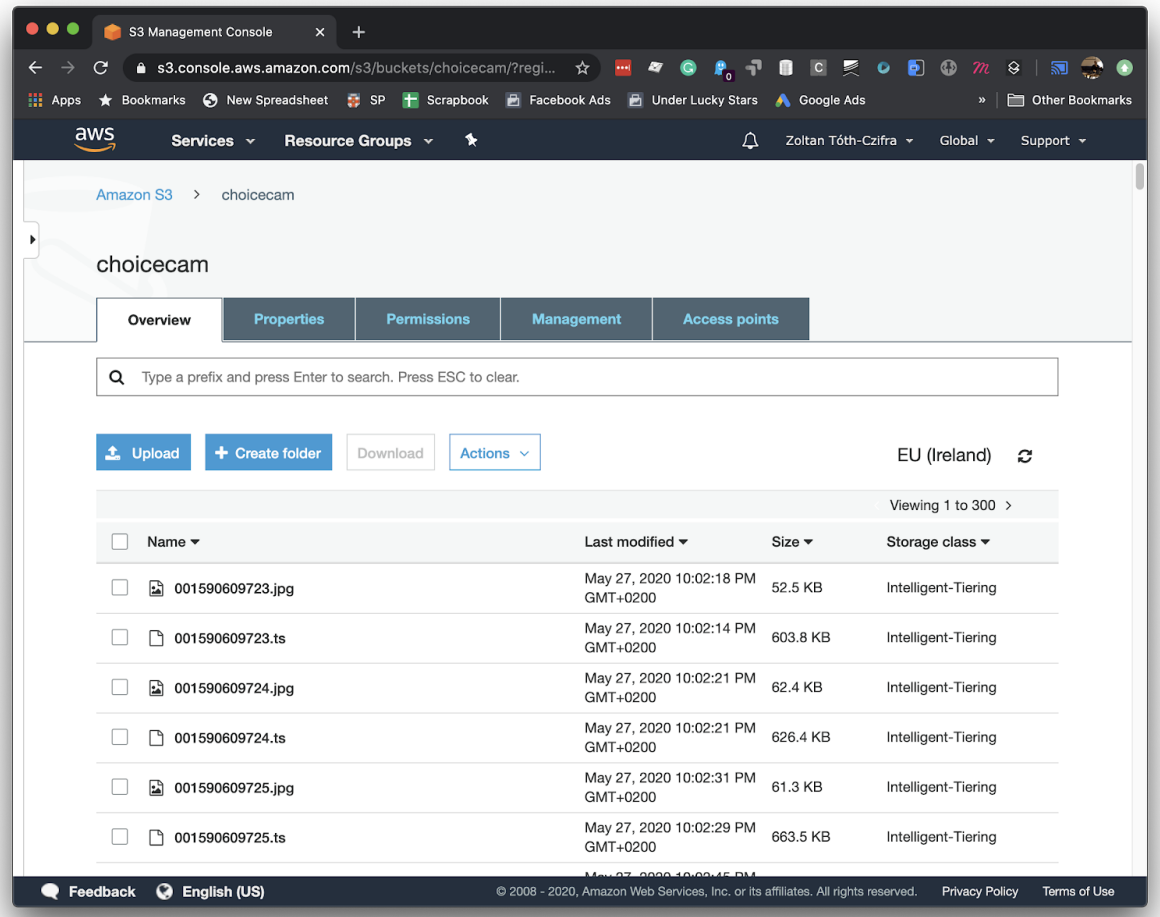


Figure 23: Accessing Choice Cam storage through Amazon Web Services Console

5.6 Improvement ideas

While the prototype is functional, it lacks some features and has some limitations. Here I list a few bullet points of improvement ideas for the future.

- Motion detection and notifications: one of the painfully missing features compared to the benchmark NestCam is the lack of motion detection coupled with push notification when the camera detects movement or sound.
- WiFi configuration: currently the only way to connect the camera to a wireless network is logging in through secure shell to the Raspberry Pi or connecting it to a monitor. Other smart home devices have the capability to share WiFi passcodes in the configuration phase from the smartphone app.

- Reset: in the current version, after the camera server is configured, there's no way to reset it to factory default and trigger the configuration phase again, except for deleting configuration files manually from it. There are reset buttons available for Raspberry Pi that can be hooked up to the motherboard and a reset feature implemented.
- Optimize storage: currently a continuous stream of video is stored, even when there are no events in the video feed. This could be optimized to only store footage that has some motion or sound. The segment list file can also be compressed (e.g. by storing ranges and not individual segments).
- Infrared camera: The current camera hardware does not provide night vision. There are Raspberry Pi compatible cameras out there with infrared sensors.
- Improve security: As discussed in the *5.3 Security* section, there are some security risks that exist in the project. These can be fixed.
- Multiple cameras or multiple clients: in its current form, there can only be one client configured for one camera server. This could be improved by allowing the configuration of several clients for the same camera, or several cameras for a client.

6 Discussion

The goal of the project has been demonstrating the separation of a service requiring cloud resources (storage in particular) from its infrastructure through a real-life working example and providing the user with the choice of where to store or process their personal data. As shown in the *5 Results* section, the proposed architecture is viable for a home security camera.

However, the concept is not limited to a single application. It is also not limited to file (blob) storage, and can be extended to relational databases, key-value databases and other types of cloud infrastructure. Below we will discuss some of the popular types of cloud services and how they could benefit from this new type of architecture.

6.1 Applicability elsewhere

6.1.1 Social networks

Imagine signing up to Facebook and being presented the following screen (Figure 24).

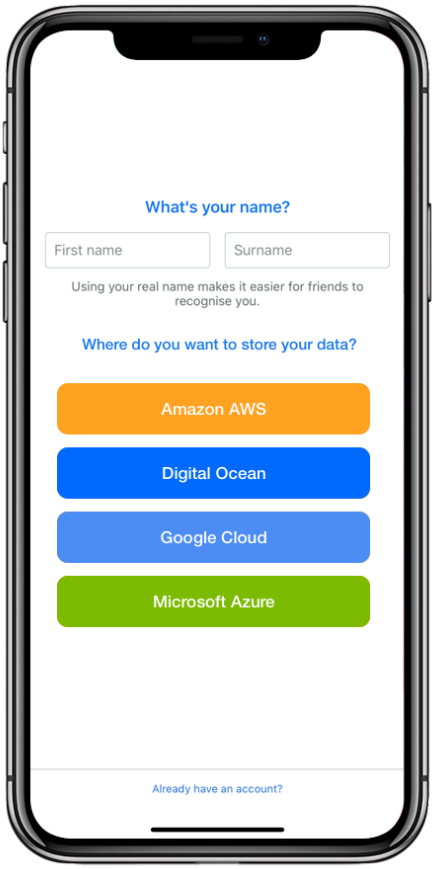


Figure 24: Facebook sign-up with choice of storage

Various distributed online social networks (DOSNs) presented in the *3 Literature Review* section provide working examples of how personal data can be dispersed over a number of loosely connected servers. Keeping parts of the service central (e.g. friend search, generating news feed, advertisement, etc) solves most of the issues that DOSNs face: identity management, working business model, content moderation, etc. In this scenario the social network pieces together its content (e.g. photos, videos, status updates) from several storage providers. It stores metadata centrally and perhaps even caches content, but the ultimate control over the data remains with the user.

6.1.2 Email

Email was designed to be a distributed service from the ground up. Similar to federated Dweb services, users choose a service provider which in turn relays email through the SMTP servers of other providers [68]. By the end of the 2010s, the email provider market however consolidated and now it is dominated by Google's Gmail service [69]. While decoupling the storage infrastructure from the service is certainly an option for email services as well, this,

unfortunately, does little to improve portability: email addresses are still tightly coupled with the chosen service provider.

6.1.3 Personal data backup

Personal data backup services such as Dropbox [70], iCloud [71], Google Drive [72] have become popular due to their ease of use and the growth of the average user's data footprint (e.g. photos taken with smartphones). However, they pose privacy risks because the providers have access to the data at rest. This was made obvious during the 2014 iCloud data breach [73] that exposed nude photos of several celebrities. Another issue is portability: it's not trivial to move from one backup provider to another one, so users might feel "locked in" through their data.

Separating the infrastructure from the service is an easily applicable concept here. The user chooses a storage provider they trust, while the service provider (Apple, Dropbox, Google…) provides seamless synchronization, indexing, versioning of data.

6.1.3 Wearables and medical devices

Some of the most sensitive kind of personal information is medical data. With the increasing popularity of wearable hardware (Apple Watch, FitBit, etc) there's also increasing concern of privacy. Since the data typically remains personal in that it requires no sharing with others, it's easy to add the choice of storage infrastructure to wearable devices.

6.2 Data portability and interoperability

Separating the infrastructure from the service and giving control to the user over their data potentially enables data portability - moving to another service together with all your data effortlessly. Controlling the storage infrastructure means that the personal data of the user cannot be held hostage.

One challenge is the structure of the data: service providers will use their own proprietary formats, database structures, etc. to store the personal data. This was seen with the prototype Choice Cam as well: segments, segment lists, encrypted thumbnails are all service-specific formats (see *5.2 Prototype*). Moving e.g. from Google Drive to Dropbox might be challenging because of the different data formats or structures used by both. However, it's not difficult to imagine the existence of services on the market that, given access to the complete data the user now controls, can "translate" from one to another. It's also easy to envision that services themselves (Dropbox in this case) would reverse-engineer data formats of their competitors and offer translating them, while keeping the personal data on the storage chosen by the user.

It does not stop there. Access to the raw and complete data enables a plethora of different services - paid or free, proprietary or open-source - that analyze, visualize, clean, search, and share the data, further increasing the visibility, utility and control of the data subject.

6.3 Risks

The user's full control over their data comes with several risks. The user can alter or delete the data on the storage provider, breaking the service that relies on that data. Service providers would need to add data integrity checks (e.g. by storing hashes of pieces of data) and handle missing or compromised data gracefully.

Since the user has full control over the data storage, there's a risk of data breach. Service providers who own and control the storage infrastructure also have better control over its security. They might be able to detect and prevent large-scale data dumps. With raw access to the data there is perhaps a higher risk. Nevertheless, storage providers themselves can develop and offer improved security services to mitigate this.

6.4 User experience

For a user with average technical skills the user experience of Choice Cam is far from ideal. In order to make the service work, they need to research, choose and sign up to an infrastructure provider, create storage space (bucket, container, space in different nomenclatures) and manage the credentials. This is much more than what can be expected from an average user.

However, in the hypothetical scenario that the proposed architecture becomes more wide-spread, the user experience is bound to improve. Storage providers can streamline their signup experience, share credentials through QR codes, and abstract other technical details from the user.

6.5 Legal enforcement

It's hard to imagine that tech giants such as Google or Facebook would voluntarily give up on their ownership of (part of) the infrastructure of the services they offer. But it's not impossible to envision new kinds of regulations in the future that get them to do that.

General Data Protection Regulation (GDPR) [74] came into force in 2018 in the European Union. It's perhaps the most widely applied data protection law in the world, and it contains interesting legal precedents of mandating where and how organizations should store personal data. For example:

- Regulating how long the data can be stored: "*[...] personal data may be stored for longer periods insofar as the personal data will be processed solely for archiving purposes in the public interest [...]*" (Article 5, section 1., paragraph (e))
- Regulating that the data needs to be stored and processed securely: "*processed in a manner that ensures appropriate security of the personal data*" (Article 5, section 1., paragraph (f))
- Establishing the right to access the data by the data subject: "*The data subject shall have the right to [...] access to the personal data*" Article 15, section 1.)
- Establishing the right to data erasure (or "be forgotten): "*The data subject shall have the right [to] the erasure of personal data*" (Article 17, section 1.)
- Establishing the right to data portability: "*The data subject shall have the right to receive the personal data [...] in a structured, commonly used and machine-readable format and have the right to transmit those data to another controller*" (Article 20, section 1.)

Perhaps built on GDPR, new regulations in the future might mandate separating infrastructure that processes and stores personal data from the service, and they might introduce the right to choose the vendor. While legal analysis is outside of the scope of this project, the proposed architecture in this paper promotes the right to data portability, erasure and access in a natural and elegant way.

The institutional separation of duties and powers has been a helpful idea in Western civilizations. For instance, the constitutional separation of state powers is the underpinning of modern democracies. The independence of central banks ensures the stability of monetary systems. It might be a concept applicable and beneficial to online services too.

As discussed in the *6.1 Applicability elsewhere* section, it has some technical implications and difficulties right now, but as technology advances (faster network speeds, better data replication technologies, etc), this becomes less and less of an issue. I don't see any fundamental technical obstacle that would make this impossible to accomplish.

7 Conclusion

In the paper I proposed a new type of online service architecture that has the potential to improve privacy and control over personal data. Through an online survey I verified that the proposal would meet popular demand and demonstrated the concept through a working prototype of a home security camera and smartphone application.

I conclude that the new type of architecture is viable and it does have the potential to improve privacy and control over personal data. I hope that we will see services in the future what employ a similar architecture. Assumptions are meant to be broken and I am glad I helped break this one.

8 Reference List

1. Facebook, Inc. Facebook [Internet]. [cited 12 Jun 2020]. Available: https://www.facebook.com/
2. Facebook, Inc. Instagram [Internet]. [cited 12 Jun 2020]. Available: https://www.instagram.com/
3. Google LLC. Google [Internet]. [cited 12 Jun 2020]. Available: https://www.google.com
4. Greenwald G, The Guardian. NSA collecting phone records of millions of Verizon customers daily [Internet]. 6 Jun 2013 [cited 18 Dec 2019]. Available: https://www.theguardian.com/world/2013/jun/06/nsa-phone-records-verizon-court-order
5. Cadwalladr C, Graham-Harrison E, The Guardian. Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach. In: Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach [Internet]. 17 Mar 2018 [cited 18 Dec 2019]. Available: https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election
6. National Highway Traffic Safety Administration, Federal Highway Administration. MOTOR VEHICLE TRAFFIC FATALITIES & FATALITY RATE: 1899 - 2003 [Internet]. National Highway Traffic Safety Administration; 2014. Available: https://web.archive.org/web/20110921222129/http://www.saferoads.org/federal/2004/TrafficFatalities1899-2003.pdf
7. Microsoft Corporation. Outlook [Internet]. [cited 12 Jun 2020]. Available: https://outlook.live.com/
8. Nest Cam Indoor - Home Security Camera - Google Store [Internet]. [cited 13 Jan 2020]. Available: https://store.google.com/us/product/nest_cam
9. Google, Inc. Google to Acquire Nest - Investor Relations - Alphabet. In: Google to Acquire Nest [Internet]. 13 Jan 2014 [cited 13 Jan 2020]. Available: https://abc.xyz/investor/news/releases/2014/0113/
10. Corbyn Z. Decentralisation: the next big step for the world wide web. The Guardian. 8 Aug 2018.
11. Chowdhury SR, Roy AR, Shaikh M, Daudjee K. A taxonomy of decentralized online social networks. Peer-to-Peer Netw Appl. 2015;8: 367–383. doi:10.1007/s12083-014-0258-2
12. Buchegger S, Schiöberg D, Vu L-H, Datta A. PeerSoN: P2P social networking: Early experiences and insights. Proceedings of the Second ACM EuroSys Workshop on Social Network Systems - SNS ’09. New York, New York, USA: ACM Press; 2009. pp. 46–52.

doi:10.1145/1578002.1578010

13. Cutillo LA, Molva R, Strufe T. Safebook: Feasibility of transitive cooperation for privacy on a decentralized social network. 2009 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks & Workshops. IEEE; 2009. pp. 1–6. doi:10.1109/WOWMOM.2009.5282446
14. Sharma R, Datta A. SuperNova: Super-peers based architecture for decentralized online social networks. 2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012). IEEE; 2012. pp. 1–10. doi:10.1109/COMSNETS.2012.6151349
15. Seong S-W, Seo J, Nasielski M, Sengupta D, Hangal S, Teh SK, et al. PrPl: A decentralized social networking infrastructure. Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services Social Networks and Beyond - MCS ’10. New York, New York, USA: ACM Press; 2010. pp. 1–8. doi:10.1145/1810931.1810939
16. Shakimov A, Lim H, Caceres R, Cox LP, Li K, Dongtao Liu, et al. Vis-à-Vis: Privacy-preserving online social networking via Virtual Individual Servers. 2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011). IEEE; 2011. pp. 1–10. doi:10.1109/COMSNETS.2011.5716497
17. Bielenberg A, Helm L, Gentilucci A, Stefanescu D, Honggang Zhang. The growth of Diaspora - A decentralized online social network in the wild. 2012 Proceedings IEEE INFOCOM Workshops. IEEE; 2012. pp. 13–18. doi:10.1109/INFCOMW.2012.6193476
18. Zignani M, Quadri C, Gaito S, Cherifi H, Rossi GP. The footprints of a “mastodon”: how a decentralized architecture influences online social relationships. IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE; 2019. pp. 472–477. doi:10.1109/INFCOMW.2019.8845221
19. Follow the “Mastodon”- Structure and Evolution of a Decentralized Online Social Network.pdf.
20. Matrix Specification [Internet]. [cited 11 Jun 2020]. Available: https://matrix.org/docs/spec/
21. Raman A, Joglekar S, Cristofaro ED, Sastry N, Tyson G. Challenges in the decentralised web: the mastodon case. Proceedings of the Internet Measurement Conference on - IMC ’19. New York, New York, USA: ACM Press; 2019. pp. 217–229. doi:10.1145/3355369.3355572
22. Bahri L, Carminati B, Ferrari E. Decentralized privacy preserving services for Online Social Networks. Online Social Networks and Media. 2018;6: 18–25. doi:10.1016/j.osnem.2018.02.001
23. Lanier J. Who Owns The Future? Penguin; 2014.

24. SurveyMonkey Inc. Sample Size Calculator: Understanding Sample Sizes | SurveyMonkey [Internet]. [cited 12 Jun 2020]. Available: https://www.surveymonkey.com/mp/sample-size-calculator/

25. Arduino s.r.l. Arduino official website [Internet]. [cited 12 Jun 2020]. Available: https://www.arduino.cc/

26. Raspberry Pi Foundation. Raspberry Pi official website [Internet]. [cited 12 Jun 2020]. Available: https://www.raspberrypi.org/

27. Sketch B.V. Sketch official website [Internet]. [cited 13 Jun 2020]. Available: https://www.sketch.com/

28. FFMPEG website [Internet]. [cited 13 Jun 2020]. Available: https://ffmpeg.org/

29. Facebook Inc. React Native official website [Internet]. [cited 13 Jun 2020]. Available: https://reactnative.dev/

30. Ecma International. ECMAScript® 2019 Language Specification, section 6.1.4 The String Type [Internet]. [cited 13 Jun 2020]. Available: https://www.ecma-international.org/ecma-262/10.0/index.html#sec-ecmascript-language-types-string-type

31. Agarwal R, Umphress D. Extreme programming for a single person team. Proceedings of the 46th Annual Southeast Regional Conference on XX - ACM-SE 46. New York, New York, USA: ACM Press; 2008. p. 82. doi:10.1145/1593105.1593127

32. Scott Chacon. Git official website [Internet]. [cited 13 Jun 2020]. Available: https://git-scm.com/

33. JetBrains s.r.o. PhpStorm official website [Internet]. [cited 13 Jun 2020]. Available: https://www.jetbrains.com/phpstorm/

34. JetBrains s.r.o. PyCharm official website [Internet]. [cited 13 Jun 2020]. Available: https://www.jetbrains.com/pycharm/?fromMenu

35. Apple, Inc. Xcode official website [Internet]. [cited 13 Jun 2020]. Available: https://developer.apple.com/xcode/

36. Apple, Inc. RFC 8216 - HTTP Live Streaming [Internet]. Aug 2017 [cited 19 May 2020]. Available: https://tools.ietf.org/html/rfc8216

37. Bitmovin, Inc. Bitmovin Video Developer Report 2019. Bitmovin, Inc.; 2019.

38. International Telecommunication Union. H.264 (06/2019). 2019.

39. National Institute of Standards and Technology. ADVANCED ENCRYPTION STANDARD (AES). 2001.

40. RTFM, Inc. RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2

[Internet]. Aug 2008 [cited 13 Jun 2020]. Available: https://tools.ietf.org/html/rfc5246

41. Dwayne C. Litzenberger. pycrypto. Dwayne C. Litzenberger; 2013.
42. Lincoln Loop. qrcode. Lincoln Loop; 2019.
43. Barker EB, Dang QH. Recommendation for Key Management Part 3: Application-Specific Key Management Guidance. Gaithersburg, MD: National Institute of Standards and Technology; 2015 Jan. doi:10.6028/NIST.SP.800-57Pt3r1
44. International Telecommunication Union. X.680 (08/2015). 2015.
45. Microsoft Corporation, Hewlett-Packard Company. Simple Service Discovery Protocol/1.0 [Internet]. 21 Jul 1999 [cited 20 May 2020]. Available: https://tools.ietf.org/id/draft-cai-ssdp-v1-02.txt
46. Moshi Binyamini. ssdpy. Moshi Binyamini; 2020.
47. Amazon Web Services. boto3. Amazon Web Services; 2020.
48. FFmpeg contributors. FFmpeg. FFmpeg contributors; 2020.
49. Martin RC. Agile Software Development. 1st Edition. Upper Saddle River, N.J: Prentice Hall; 2003. p. 529.
50. 650 Industries, Inc. expo-barcode-scanner. 650 Industries, Inc.; 2020.
51. Tradle, Inc. react-native-udp. Tradle, Inc.; 2020.
52. Amazon Web Services, Inc. Amazon Simple Storage Service (S3) [Internet]. [cited 13 Jun 2020]. Available: https://aws.amazon.com/s3/
53. Google Ireland Limited. Google Cloud Cloud Storage [Internet]. [cited 13 Jun 2020]. Available: https://cloud.google.com/storage
54. Microsoft Corporation. Microsoft Azure Azure Blob Storage [Internet]. [cited 13 Jun 2020]. Available: https://azure.microsoft.com/en-us/services/storage/blobs/
55. DigitalOcean, LLC. Digital Ocean Spaces Object Storage [Internet]. [cited 13 Jun 2020]. Available: https://www.digitalocean.com/products/spaces/
56. tectiv3. react-native-aes-crypto. tectiv3; 2020.
57. Toth-Czifra Z. 128-bit encryption, handling binary data · tcz/react-native-aes@7694604 [Internet]. [cited 13 Jun 2020]. Available: https://github.com/tcz/react-native-aes/commit/76946044a31fdaf73eb37a9370960d7ea98fa971
58. Sam Saffron. react-native-rsa-native. Sam Saffron; 2020.

59. 650 Industries, Inc. expo-secure-store. 650 Industries, Inc.; 2020.

60. Futurepress. react-native-static-server. Futurepress; 2019.

61. 650 Industries, Inc. expo-file-system. 650 Industries, Inc.; 2020.

62. Google, Inc. Google Nest Indoor Product Page [Internet]. [cited 13 Jun 2020]. Available: https://store.google.com/es/category/connected_home?hl=es-ES&GoogleNest&utm_source=nest_redirect&utm_medium=google_oo&utm_campaign=GS102776&utm_term=control

63. Google, Inc. Nest Aware Pricing [Internet]. [cited 13 Jun 2020]. Available: https://store.google.com/product/nest_aware

64. Amazon Web Services, Inc. Amazon S3 Simple Storage Service Pricing [Internet]. [cited 13 Jun 2020]. Available: https://aws.amazon.com/s3/pricing/

65. Google Ireland Limited. Google Cloud Storage pricing [Internet]. [cited 13 Jun 2020]. Available: https://cloud.google.com/storage/pricing

66. Microsoft Corporation. Microsoft Azure Storage Blobs Pricing [Internet]. [cited 13 Jun 2020]. Available: https://azure.microsoft.com/en-us/pricing/details/storage/blobs/

67. DigitalOcean, LLC. Digital Ocean Pricing [Internet]. [cited 13 Jun 2020]. Available: https://www.digitalocean.com/pricing/

68. J. Klensin. RFC 5321 - Simple Mail Transfer Protocol [Internet]. Oct 2008 [cited 13 Jun 2020]. Available: https://tools.ietf.org/html/rfc5321

69. Litmus Software, Inc. Email Client Market Share and Popularity - May 2020 [Internet]. Jun 2020 [cited 13 Jun 2020]. Available: https://emailclientmarketshare.com/

70. Dropbox, Inc. Dropbox official website [Internet]. [cited 13 Jun 2020]. Available: https://www.dropbox.com

71. Apple, Inc. Apple iCloud official website [Internet]. [cited 13 Jun 2020]. Available: https://www.icloud.com/

72. Google, Inc. Google Drive official website [Internet]. [cited 13 Jun 2020]. Available: https://www.google.com/drive/

73. Kelion L. Apple toughens iCloud security after celebrity breach - BBC News. BBC News. 17 Sep 2014.

74. European Council. REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) [Internet]. 2016/679 Apr 27, 2016. Available:

https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN

75. Toth-Czifra Z. How I turned my old rotary phone into a virtual assistant [Internet]. 15 Nov 2018 [cited 13 Jun 2020]. Available: https://medium.com/@tcz/how-i-turned-my-old-rotary-phone-into-a-virtual-assistant-24f35ca9884f

9 Appendix

9.1 Glossary of Acronyms and Abbreviations

While I tried to indicate the full name of a concept, technology, protocol in the paper at least once before relying on acronyms and abbreviations, the reader might still find it useful to see a list of these.

AES-128	Advanced Encryption Standard, 128 bit key length
API	Application programming interface
App	Application
ASN.1	Abstract Syntax Notation One
AWS	Amazon Web Services
CBC	Cipher block chaining
CPU	Central Processing Unit
DER	Distinguished Encoding Rules
DOSN	Distributed Online Social Network
Dweb	Distributed Web, Decentralized Web
FOV	Field of View
GDPR	General Data Protection Regulation
GPU	Graphics processing unit
H.264	Advanced Video Coding
HLS	HTTP Live Streaming
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated development environment
IP	Internet Protocol
IRC	Internet Relay Chat
IT	Information Technology
IV	Initialization Vector

JPEG	Joint Photographic Experts Group (file format)
JSX	JavaScript XML
M3U8/M3U	Moving Picture Experts Group Audio Layer 3 Uniform Resource Locator
MP	megapixel
MPEG-2 TS	Moving Picture Experts Group Transport Stream
OS	Operating system
OSN	Online social media
PEM	Privacy-Enhanced Mail
PKCS#1	Public-Key Cryptography Standards, 1st family of standards
QR code	Quick Response code
RAM	Random access memory
S3	Simple Storage Service
SMTP	Simple Mail Transfer Protocol
SSDP	Simple Service Discovery Protocol
SSH	Secure shell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UK	United Kingdom
US	United States of America
USD	United States Dollar
USN	Unique Service Name
UTF-16	Unicode Transformation Format, 16-bit
WiFi	Wireless network protocols under IEEE 802.11
WWW	World wide web
XMPP	Extensible Messaging and Presence Protocol
XP	Extreme Programming

9.2 Project plan

Below is the updated project plan from the Preliminary Project Report.

Task	Duration	Start date	End date	Completeness
Literature review	21 days	November 2019	February 2020	100%
Feasibility study	7 days	December 2019	December 2019	100%
Hardware research	3 days	December 2019	December 2019	100%
Develop survey	3 days	December 2019	December 2019	100%
Obtain survey results	14 days	December 2019	December 2019	100%
Procure hardware	1 day	December 2019	December 2019	100%
Camera proof of concept	7 days	December 2019	January 2020	100%
Implementing Video Capture Software	7 days	April 2020	May 2020	100%
Implementing Cloud Syncing Software	7 days	April 2020	May 2020	100%
Implementing Camera API	7 days	April 2020	May 2020	100%
Implementing Smart Phone Application - Application Basics and Camera Set Up	14 days	April 2020	May 2020	100%
Implementing Smart Phone Application - Video Viewing	14 days	April 2020	May 2020	100%
Writing Final Project Report	14 days	May 2020	June 2020	100%

9.3 Survey answers

Due to the wideness of the table I split the answers to several tables. The responses can be identified by their time stamps. The auxiliary files include an Excel sheet with the response.

Timestamp	How concerned are you about online privacy in relation to large tech companies (Google, Facebook, Amazon, Apple, etc)?	How much do you trust large tech companies (Google, Facebook, Amazon, Apple, etc) to handle your private data responsibly?	How much do you trust large tech companies to completely erase all your personal data when you remove your account?
12/19/2019 3:35:19	2	4	1
12/19/2019 3:52:58	4	2	4
12/19/2019 4:02:52	3	4	3
12/19/2019 4:17:49	1	1	1
12/19/2019 4:27:41	5	1	1
12/19/2019 4:47:24	3	2	2
12/19/2019 4:54:13	3	2	1
12/19/2019 5:39:17	3	2	1
12/19/2019 5:40:44	4	4	3
12/19/2019 5:49:22	4	3	1
12/19/2019 6:02:42	3	3	2
12/19/2019 6:07:17	4	2	1
12/19/2019 6:58:08	4	1	1
12/19/2019 7:02:56	2	3	4
12/19/2019 7:27:35	5	1	1
12/19/2019 10:30:08	5	1	1
12/19/2019 10:33:26	5	1	2
12/19/2019 10:40:21	4	5	2
12/19/2019 12:29:13	4	1	1
12/19/2019 13:41:40	4	2	2
12/19/2019 14:07:43	5	2	1

12/19/2019 14:13:29	4	4	4
12/19/2019 15:13:22	3	3	2
12/19/2019 16:01:07	4	2	1
12/19/2019 16:51:14	4	2	1
12/19/2019 16:57:24	5	1	1
12/19/2019 20:00:54	5	5	4
12/20/2019 2:31:36	5	3	2
12/21/2019 2:38:23	4	2	2
12/21/2019 3:53:32	4	2	1
12/21/2019 8:37:02	2	3	4
12/21/2019 16:26:28	5	5	5
12/21/2019 21:08:28	3	3	4
12/22/2019 5:07:52	5	4	3

Timestamp	Have you ever deleted or considered deleting any of your social media or cloud storage (email, files) accounts due to privacy concerns?	What steps do you think need to be taken to improve online privacy in relation to large tech companies?	Which of these services are you using actively?
12/19/2019 3:35:19	Did not consider	More or better regulations by the government, Technical solutions that improve privacy	Google Search, Facebook, Gmail
12/19/2019 3:52:58	Did not consider	Technical solutions that improve privacy, Tech companies should self-regulate	Google Search, Instagram, Reddit, Snapchat, Gmail, Amazon (for shopping)

12/19/2019 4:02:52	Did not consider	More or better regulations by the government	Facebook, Instagram, Gmail
12/19/2019 4:17:49	Did delete	More or better regulations by the government, stop UK surveillence on the general public	Google Search, Facebook, Gmail
12/19/2019 4:27:41	Did delete	Users should stop using products from large tech companies	Google Search, Snapchat, Gmail
12/19/2019 4:47:24	Did delete	More or better regulations by the government, Technical solutions that improve privacy	Google Search, Twitter, Instagram, Gmail
12/19/2019 4:54:13	Did consider but did not do it (yet)	Technical solutions that improve privacy, Tech companies should self-regulate	Google Search, Twitter, Instagram, Gmail, Amazon (for shopping)
12/19/2019 5:39:17	Did delete	More or better regulations by the government, Technical solutions that improve privacy	Google Search, Twitter, Instagram, Pinterest, Reddit, Gmail, Hotmail or Outlook.com, Amazon (for shopping), Nest Cam
12/19/2019 5:40:44	Did not consider	Technical solutions that improve privacy, Tech companies should self-regulate	Google Search, Facebook, Instagram, Gmail

12/19/2019 5:49:22	Did delete	More or better regulations by the government	Google Search, Twitter, Instagram, Gmail, Amazon (for shopping)
12/19/2019 6:02:42	Did not consider	Technical solutions that improve privacy, Tech companies should self-regulate	Google Search, Facebook, Twitter, Instagram, Gmail, Amazon (for shopping)
12/19/2019 6:07:17	Did consider but did not do it (yet)	More or better regulations by the government	Google Search, Facebook, Instagram, Reddit, Gmail, Amazon (for shopping)
12/19/2019 6:58:08	Did consider but did not do it (yet)	More or better regulations by the government, Technical solutions that improve privacy	Google Search, Instagram, Snapchat, Gmail, Amazon (for shopping)
12/19/2019 7:02:56	Did not consider	More or better regulations by the government	Google Search, Instagram, Gmail, Hotmail or Outlook.com
12/19/2019 7:27:35	Did delete	Technical solutions that improve privacy	Google Search, Facebook, Pinterest, Gmail, Hotmail or Outlook.com
12/19/2019 10:30:08	Did consider but did not do it (yet)	More or better regulations by the government, Technical solutions that improve privacy, Tech companies should self-regulate	Google Search, Facebook, Instagram, Reddit, Snapchat, Hotmail or Outlook.com

12/19/2019 10:33:26	Did consider but did not do it (yet)	More or better regulations by the government, Users should stop using products from large tech companies	Google Search, Instagram, Gmail, Hotmail or Outlook.com, Amazon (for shopping)
12/19/2019 10:40:21	Did consider but did not do it (yet)	More or better regulations by the government	Google Search, Instagram, Snapchat, Gmail, Hotmail or Outlook.com, Amazon (for shopping)
12/19/2019 12:29:13	Did not consider	More or better regulations by the government, Technical solutions that improve privacy, Tech companies should self-regulate	Facebook, Twitter, Hotmail or Outlook.com, Amazon (for shopping)
12/19/2019 13:41:40	Did consider but did not do it (yet)	More or better regulations by the government	Google Search, Facebook, Twitter, Instagram, Pinterest, Snapchat, Hotmail or Outlook.com, Amazon (for shopping)
12/19/2019 14:07:43	Did consider but did not do it (yet)	Technical solutions that improve privacy, Tech companies should self-regulate	Google Search, Facebook, Instagram, Reddit, Gmail, Hotmail or Outlook.com, Amazon (for shopping)
12/19/2019 14:13:29	Did not consider	More or better regulations by the government	Google Search, Facebook, Gmail, Hotmail or Outlook.com

12/19/2019 15:13:22	Did not consider	More or better regulations by the government, Tech companies should self-regulate	Google Search, Facebook, Twitter, Instagram, Pinterest, Gmail
12/19/2019 16:01:07	Did consider but did not do it (yet)	More or better regulations by the government	Instagram, Amazon (for shopping)
12/19/2019 16:51:14	Did consider but did not do it (yet)	More or better regulations by the government	Google Search, Facebook, Gmail, Hotmail or Outlook.com, Amazon (for shopping)
12/19/2019 16:57:24	Did consider but did not do it (yet)	More or better regulations by the government	Google Search, Facebook, Instagram
12/19/2019 20:00:54	Did consider but did not do it (yet)	Technical solutions that improve privacy	Facebook
12/20/2019 2:31:36	Did consider but did not do it (yet)	More or better regulations by the government	Facebook
12/21/2019 2:38:23	Did consider but did not do it (yet)	More or better regulations by the government, Technical solutions that improve privacy, Tech companies should self-regulate	Google Search, Facebook, Gmail, Hotmail or Outlook.com, Amazon (for shopping)
12/21/2019 3:53:32	Did consider but did not do it (yet)	More or better regulations by the government, Technical solutions that improve privacy	Google Search, Instagram, Pinterest, Snapchat, Gmail, Hotmail or Outlook.com, Amazon (for shopping)

12/21/2019 8:37:02	Did not consider	More or better regulations by the government, Technical solutions that improve privacy, Tech companies should self-regulate	Google Search, Facebook, Instagram, Snapchat, Gmail
12/21/2019 16:26:28	Did delete	More or better regulations by the government	Facebook
12/21/2019 21:08:28	Did not consider	Tech companies should self-regulate	Facebook
12/22/2019 5:07:52	Did not consider	More or better regulations by the government	Facebook

Timestamp	Do you own any home security device (e.g. camera) that uploads data to the cloud (e.g. a video feed)?	Do you use any home assistant device like Google Home or Amazon Echo?	How complete idea you have about what data is collected and kept about you by the large tech companies?
12/19/2019 3:35:19	No	No	2
12/19/2019 3:52:58	No	Yes	3
12/19/2019 4:02:52	No	Yes	1
12/19/2019 4:17:49	No	No	1
12/19/2019 4:27:41	No	No	3
12/19/2019 4:47:24	No	No	2
12/19/2019 4:54:13	No	No	2

12/19/2019 5:39:17	Yes	Yes	5
12/19/2019 5:40:44	No	No	4
12/19/2019 5:49:22	No	No	3
12/19/2019 6:02:42	I don't know	No	2
12/19/2019 6:07:17	No	Yes	1
12/19/2019 6:58:08	No	No	2
12/19/2019 7:02:56	No	Yes	3
12/19/2019 7:27:35	No	No	1
12/19/2019 10:30:08	I don't know	No	1
12/19/2019 10:33:26	Yes	No	1
12/19/2019 10:40:21	No	No	4
12/19/2019 12:29:13	No	No	1
12/19/2019 13:41:40	No	No	3
12/19/2019 14:07:43	Yes	Yes	3
12/19/2019 14:13:29	No	Yes	3
12/19/2019 15:13:22	No	No	4
12/19/2019 16:01:07	No	No	3
12/19/2019 16:51:14	No	Yes	3
12/19/2019 16:57:24	No	No	1
12/19/2019 20:00:54	Yes	No	4
12/20/2019 2:31:36	No	No	2
12/21/2019 2:38:23	No	Yes	4

12/21/2019 3:53:32	No	No	2
12/21/2019 8:37:02	No	No	1
12/21/2019 16:26:28	No	No	4
12/21/2019 21:08:28	No	No	3
12/22/2019 5:07:52	No	No	5

Timestamp	How complete idea you have about where and how the data collected about you stored by the large tech companies?	Would you be interested in seeing the complete raw data stored about you by large tech companies?	Would you feel more comfortable using an online service if you could select where (in which country) your data is stored?
12/19/2019 3:35:19	2	No	Yes
12/19/2019 3:52:58	1	I don't know	No
12/19/2019 4:02:52	3	Yes	Yes
12/19/2019 4:17:49	1	Yes	Yes
12/19/2019 4:27:41	3	Yes	Yes
12/19/2019 4:47:24	2	Yes	Yes
12/19/2019 4:54:13	2	Yes	I don't know
12/19/2019 5:39:17	5	No	Yes
12/19/2019 5:40:44	4	Yes	I don't know
12/19/2019 5:49:22	1	Yes	I don't know
12/19/2019 6:02:42	2	No	I don't know

12/19/2019 6:07:17	1	I don't know	Yes
12/19/2019 6:58:08	1	Yes	No
12/19/2019 7:02:56	2	No	I don't know
12/19/2019 7:27:35	1	Yes	Yes
12/19/2019 10:30:08	1	Yes	Yes
12/19/2019 10:33:26	1	Yes	No
12/19/2019 10:40:21	4	No	Yes
12/19/2019 12:29:13	1	Yes	I don't know
12/19/2019 13:41:40	2	Yes	I don't know
12/19/2019 14:07:43	3	Yes	Yes
12/19/2019 14:13:29	3	I don't know	I don't know
12/19/2019 15:13:22	3	Yes	I don't know
12/19/2019 16:01:07	2	No	I don't know
12/19/2019 16:51:14	2	Yes	Yes
12/19/2019 16:57:24	1	I don't know	I don't know
12/19/2019 20:00:54	5	Yes	No
12/20/2019 2:31:36	1	I don't know	Yes
12/21/2019 2:38:23	2	Yes	Yes
12/21/2019 3:53:32	2	Yes	No
12/21/2019 8:37:02	1	Yes	Yes
12/21/2019 16:26:28	4	Yes	Yes
12/21/2019 21:08:28	4	No	No

12/22/2019 5:07:52	4	Yes	Yes

Timestamp	Would you feel more comfortable using an online service if you could fully see your stored data at all times?	Would you feel more comfortable using an online service if the company providing the service (e.g. Gmail, Facebook) was different from the one storing your personal data involved in using the service (e.g. your emails or photos)?	Would you feel more comfortable using an online service if the company providing the service (e.g. Gmail, Facebook) if you could choose from a list of providers that will store your data involved in using the service (e.g. your emails or photos)?
12/19/2019 3:35:19	Yes	Yes	Yes
12/19/2019 3:52:58	Yes	I don't know	I don't know
12/19/2019 4:02:52	Yes	I don't know	Yes
12/19/2019 4:17:49	No	Yes	No
12/19/2019 4:27:41	Yes	No	No
12/19/2019 4:47:24	Yes	Yes	Yes
12/19/2019 4:54:13	Yes	Yes	Yes
12/19/2019 5:39:17	Yes	Yes	I don't know
12/19/2019 5:40:44	I don't know	No	No
12/19/2019 5:49:22	Yes	I don't know	I don't know
12/19/2019 6:02:42	Yes	Yes	Yes

12/19/2019 6:07:17	Yes	I don't know	I don't know
12/19/2019 6:58:08	Yes	No	Yes
12/19/2019 7:02:56	Yes	I don't know	Yes
12/19/2019 7:27:35	Yes	I don't know	Yes
12/19/2019 10:30:08	Yes	Yes	Yes
12/19/2019 10:33:26	No	Yes	No
12/19/2019 10:40:21	Yes	Yes	Yes
12/19/2019 12:29:13	Yes	I don't know	I don't know
12/19/2019 13:41:40	Yes	I don't know	I don't know
12/19/2019 14:07:43	Yes	Yes	Yes
12/19/2019 14:13:29	I don't know	I don't know	I don't know
12/19/2019 15:13:22	Yes	No	I don't know
12/19/2019 16:01:07	Yes	I don't know	No
12/19/2019 16:51:14	Yes	I don't know	Yes
12/19/2019 16:57:24	Yes	I don't know	I don't know
12/19/2019 20:00:54	I don't know	Yes	Yes
12/20/2019 2:31:36	Yes	No	Yes
12/21/2019 2:38:23	Yes	Yes	Yes
12/21/2019 3:53:32	Yes	No	No
12/21/2019 8:37:02	Yes	Yes	Yes
12/21/2019 16:26:28	Yes	Yes	Yes
12/21/2019 21:08:28	No	No	No

12/22/2019 5:07:52	Yes	Yes	Yes

Timestamp	What's your nationality?	What's your age?
12/19/2019 3:35:19	UK	50
12/19/2019 3:52:58	American	17
12/19/2019 4:02:52	Spanish	18
12/19/2019 4:17:49	British	31
12/19/2019 4:27:41	American	19
12/19/2019 4:47:24	Russian	21
12/19/2019 4:54:13	American	18
12/19/2019 5:39:17	Jewish	24
12/19/2019 5:40:44	Black	46
12/19/2019 5:49:22	Spanish	18
12/19/2019 6:02:42	American	21
12/19/2019 6:07:17	American	19
12/19/2019 6:58:08	United States	18
12/19/2019 7:02:56	American	14
12/19/2019 7:27:35	Catalan	54
12/19/2019 10:30:08	Welsh	18
12/19/2019 10:33:26	British	23
12/19/2019 10:40:21	British Bangladeshi	18
12/19/2019 12:29:13	British	21

12/19/2019 13:41:40	british	19
12/19/2019 14:07:43	Romanian	20
12/19/2019 14:13:29	British / English	38
12/19/2019 15:13:22	U.S.	29
12/19/2019 16:01:07	British	28
12/19/2019 16:51:14	British	53
12/19/2019 16:57:24	British	53
12/19/2019 20:00:54	Rumana	54
12/20/2019 2:31:36	British	36
12/21/2019 2:38:23	British	51
12/21/2019 3:53:32	English	17
12/21/2019 8:37:02	Scottish	50
12/21/2019 16:26:28	Spain	54
12/21/2019 21:08:28	Spanish	45
12/22/2019 5:07:52	English	49

9.4 Source code

The full source code can be found attached in the auxiliary files, but I also copy the non-binary files written by me here. The source code is also available publicly under the following URLs:

- https://github.com/tcz/choicecam-server
- https://github.com/tcz/choicecam-client

9.4.1 Camera server

providers/__init__.py

```
from .AwsProvider import AwsProvider
from .AzureProvider import AzureProvider
from .DigitalOceanProvider import DigitalOceanProvider
from .GoogleCloudProvider import GoogleCloudProvider

all_providers = {
   'aws': AwsProvider,
   'azure': AzureProvider,
   'digitalocean': DigitalOceanProvider,
   'googlecloud': GoogleCloudProvider,
}

def provider_factory(configuration):
   type = configuration['type']
   credentials = configuration['credentials']
   for provider_type, provider_class in all_providers.items():
       if provider_type == type:
           return provider_class(credentials)

   raise KeyError('No provider found for type ' + type)
```

providers/AwsProvider.py

```
import boto3
from botocore.client import Config

class AwsProvider:
   def __init__(self, credentials):
       self.bucket_name = credentials['bucket_name']
       self.region = credentials['region']
```

```
        self.key = credentials['key']
        self.secret = credentials['secret']
        self.client = None

        pass

    def credentials_correct(self):
        try:
            client = self.get_client()
            client.list_objects(Bucket=self.bucket_name)
        except:
            return False

        return True

    def sync_file(self, key, path, content_type):
        print("Uploading " + path + " to " + key)

        with open(path, 'rb') as data:
            self.get_client().put_object(Key=key, Body=data, Bucket=self.bucket_name,
                                ACL='public-read',
                                StorageClass='INTELLIGENT_TIERING',
ContentType=content_type)

    def sync_string(self, key, content, content_type):
        self.get_client().put_object(Key=key, Body=content, Bucket=self.bucket_name,
                                 ACL='public-read',
                                 StorageClass='INTELLIGENT_TIERING',
ContentType=content_type)

    def base_path(self):
        return "https://" + self.bucket_name + ".s3-" + self.region + ".amazonaws.com/"

    def get_client(self):
        if self.client is not None:
            return self.client

        self.client = boto3.client('s3',
                                   aws_access_key_id=self.key,
                                   aws_secret_access_key=self.secret,
                                   config=Config(
                                        max_pool_connections=50,
                                        region_name = self.region,
                                    ))

        return self.client
```

providers/AzureProvider.py

```
import os, uuid
from azure.storage.blob import BlobServiceClient, BlobClient, ContainerClient,
ContentSettings


class AzureProvider:
   def __init__(self, credentials):
       self.container_name = credentials['container_name']
       self.connection_string = credentials['connection_string']
       self.client = None

       pass

   def credentials_correct(self):
       try:
           client = self.get_client()
           container_client = client.get_container_client(self.container_name)
           container_client.list_blobs()
       except:
           return False

       return True

   def sync_file(self, key, path, content_type):
       with open(path, 'rb') as data:
           blob_client = self.get_client().get_blob_client(container=self.container_name,
blob=key)
           my_content_settings = ContentSettings(content_type=content_type)
           blob_client.upload_blob(data, overwrite=True,
content_settings=my_content_settings)

   def sync_string(self, key, content, content_type):
       blob_client = self.get_client().get_blob_client(container=self.container_name,
blob=key)
       my_content_settings = ContentSettings(content_type=content_type)
       blob_client.upload_blob(content, overwrite=True,
content_settings=my_content_settings)

   def base_path(self):
       return "https://" + self.container_name + ".blob.core.windows.net/" +
self.container_name + "/"

   def get_client(self):
       if self.client is not None:
```

```
            return self.client

        self.client = BlobServiceClient.from_connection_string(self.connection_string)

        return self.client
```

providers/DigitalOceanProvider.py

```
import boto3
from botocore.client import Config

class DigitalOceanProvider:
    def __init__(self, credentials):
        self.bucket_name = credentials['bucket_name']
        self.region = credentials['region']
        self.key = credentials['key']
        self.secret = credentials['secret']
        self.client = None

        pass

    def credentials_correct(self):
        try:
            client = self.get_client()
            client.list_objects(Bucket=self.bucket_name)
        except:
            return False

        return True

    def sync_file(self, key, path, content_type):
        print("Uploading " + path + " to " + key)

        with open(path, 'rb') as data:
            self.get_client().put_object(Key=key, Body=data, Bucket=self.bucket_name,
                                         ACL='public-read',
                                         ContentType=content_type)

    def sync_string(self, key, content, content_type):

        self.get_client().put_object(Key=key, Body=content, Bucket=self.bucket_name,
                                     ACL='public-read',
                                     ContentType=content_type)
```

```
    def base_path(self):
        return "https://" + self.bucket_name + "." + self.region + ".digitaloceanspaces.com/"

    def get_client(self):
        if self.client is not None:
            return self.client
        # Digital Ocean has an Amazon S3 compatible API, so we can use Boto.
        self.client = boto3.client('s3',
                                   aws_access_key_id=self.key,
                                   aws_secret_access_key=self.secret,
                                   endpoint_url='https://' + self.region +
'.digitaloceanspaces.com',
                                   config=Config(
                                        max_pool_connections=50,
                                    ))

        return self.client
```

providers/GoogleCloudProvider.py

```
import boto3
from botocore.client import Config

class GoogleCloudProvider:
    def __init__(self, credentials):
        self.bucket_name = credentials['bucket_name']
        self.key = credentials['key']
        self.secret = credentials['secret']
        self.client = None

        pass

    def credentials_correct(self):
        try:
            client = self.get_client()
            client.list_objects(Bucket=self.bucket_name)
        except:
            return False

        return True

    def sync_file(self, key, path, content_type):
        print("Uploading " + path + " to " + key)

        with open(path, 'rb') as data:
```

```
            self.get_client().put_object(Key=key, Body=data, Bucket=self.bucket_name,
                                 ACL='public-read',
                                 ContentType=content_type)

    def sync_string(self, key, content, content_type):
        self.get_client().put_object(Key=key, Body=content, Bucket=self.bucket_name,
                                  ACL='public-read',
                                  ContentType=content_type)

    def base_path(self):
        return "https://storage.cloud.google.com/" + self.bucket_name + "/"

    def get_client(self):
        if self.client is not None:
            return self.client
        # Google has an Amazon S3 compatible API, so we can use Boto.
        self.client = boto3.client('s3',
                                   aws_access_key_id=self.key,
                                   aws_secret_access_key=self.secret,
                                   endpoint_url="https://storage.googleapis.com",
                                   config=Config(
                                        max_pool_connections=50,
                                        region_name = 'auto',
                                    ))

        return self.client
```

config.py

```
import binascii
import os
import json
from providers import provider_factory

CONFIG_PATH = os.path.realpath(os.path.join(os.path.dirname(os.path.realpath(__file__)),
'config'))

PRIVATE_KEY_FILE_PATH           = os.path.join(CONFIG_PATH, 'server.key')
PUBLIC_KEY_FILE_PATH            = os.path.join(CONFIG_PATH, 'server.pub')
PUBLIC_KEY_BARE_FILE_PATH       = os.path.join(CONFIG_PATH, 'server.bare.pub')
PUBLIC_KEY_QR_CODE_FILE_PATH    = os.path.join(CONFIG_PATH, 'server.pub.png')
SYMMETRIC_KEY_FILE_PATH         = os.path.join(CONFIG_PATH, 'symmetric.key')
SYMMETRIC_KEY_INFO_FILE_PATH    = os.path.join(CONFIG_PATH, 'symmetric.keyinfo')
PROVIDER_DATA_FILE_PATH         = os.path.join(CONFIG_PATH, 'provider_data.json')
VIDEO_OUTPUT_PATH               = os.path.realpath(os.path.join(CONFIG_PATH, '../output'))
```

```
FONT_PATH                       = os.path.realpath(os.path.join(CONFIG_PATH,
'../quicksand-500.ttf'))
PLAYLIST_FILE_PATH              = os.path.join(VIDEO_OUTPUT_PATH, "playlist.m3u8")

def has_config_for_streaming():
   return os.path.isfile(SYMMETRIC_KEY_FILE_PATH) and \
          os.path.isfile(PROVIDER_DATA_FILE_PATH)

def save_config_for_streaming(config_from_client):

   with open(SYMMETRIC_KEY_FILE_PATH, 'wb') as symmetric_key_file:
       symmetric_key_file.write(binascii.unhexlify(config_from_client['symmetric_key']))

   with open(SYMMETRIC_KEY_INFO_FILE_PATH, 'w') as symmetric_key_info_file:
       symmetric_key_info_file.write(SYMMETRIC_KEY_FILE_PATH + "\n" +
SYMMETRIC_KEY_FILE_PATH)

   with open(PROVIDER_DATA_FILE_PATH, 'w') as provider_data_file:
       provider_data_file.write(json.dumps(config_from_client['provider_data']))


def provider_from_config():
   if not has_config_for_streaming():
       raise RuntimeError('No config for streaming found')

   with open(PROVIDER_DATA_FILE_PATH, 'r') as provider_data_file:
       provider_data = json.loads(provider_data_file.read())

   return provider_factory(provider_data)
```

crypto.py

```
import binascii
import os, random, struct
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

def encrypt_file(key_filename, in_filename, out_filename):
   chunksize = 64 * 1024

   with open(key_filename, 'rb') as key_file:
       key = key_file.read(16)

   iv = get_random_bytes(16)
```

```
    encryptor = AES.new(key, AES.MODE_CBC, iv)

    with open(in_filename, 'rb') as infile:
        with open(out_filename, 'wb') as outfile:
            outfile.write(binascii.hexlify(iv))
            outfile.write( b"\0\0\0\0")

            while True:
                chunk = infile.read(chunksize)
                if len(chunk) == 0:
                    break
                elif len(chunk) % 16 != 0:
                    chunk += b"\0" * (16 - len(chunk) % 16)

                outfile.write(encryptor.encrypt(chunk))
```

generate_keys.py

```
from Crypto.PublicKey import RSA
from Crypto import Random
import os
import qrcode
from config import PRIVATE_KEY_FILE_PATH, PUBLIC_KEY_FILE_PATH, PUBLIC_KEY_BARE_FILE_PATH,
PUBLIC_KEY_QR_CODE_FILE_PATH

if not os.path.isfile(PUBLIC_KEY_FILE_PATH) and \
  not os.path.isfile(PRIVATE_KEY_FILE_PATH) and \
  not os.path.isfile(PUBLIC_KEY_BARE_FILE_PATH) and \
  not os.path.isfile(PUBLIC_KEY_QR_CODE_FILE_PATH):

    random_generator = Random.new().read
    key = RSA.generate(2048, random_generator)
    private, public = key.exportKey(), key.publickey().exportKey()
    bare_public = public \
        .replace(b"\n", b"") \
        .replace(b"-----BEGIN PUBLIC KEY-----", b"") \
        .replace(b"-----END PUBLIC KEY-----", b"")

    with open(PRIVATE_KEY_FILE_PATH, 'wb') as private_file:
        private_file.write(private)
    with open(PUBLIC_KEY_FILE_PATH, 'wb') as public_file:
        public_file.write(public)
    with open(PUBLIC_KEY_BARE_FILE_PATH, 'wb') as bare_public_file:
        bare_public_file.write(bare_public)
```

```
    image = qrcode.make(bare_public)
    image.save(PUBLIC_KEY_QR_CODE_FILE_PATH)
```

run.py

```
import signal
import os
import time

from config import has_config_for_streaming
from ssdp import run as run_ssdp
from webservice import run as run_webservice
from videocapturer import run as run_videcapturer
from videouploader import run as run_videouploader

def start_config_services():
    ssdp_process_id = os.fork()
    if ssdp_process_id == 0:
        run_ssdp()
        return

    webservice_process_id = os.fork()
    if webservice_process_id == 0:
        run_webservice()
        return

    exited_process_id, exited_process_status = os.wait()
    if exited_process_id == webservice_process_id and exited_process_status == 0:
        os.kill(ssdp_process_id, signal.SIGTERM)
        time.sleep(1)
        start_services()
        return

    restart = False
    if ssdp_process_id == exited_process_id:
        os.kill(webservice_process_id, signal.SIGTERM)
        restart = True
    elif webservice_process_id == exited_process_id:
        os.kill(ssdp_process_id, signal.SIGTERM)
        restart = True

    if restart:
        time.sleep(1)
        start_config_services()
```

```
def start_streaming_services():
    capturer_process_id = os.fork()
    if capturer_process_id == 0:
        run_videcapturer()
        return

    uploader_process_id = os.fork()
    if uploader_process_id == 0:
        run_videouploader()
        return

    exited_process_id, exited_process_status = os.wait()

    if capturer_process_id == exited_process_id:
        os.kill(uploader_process_id, signal.SIGTERM)
    elif uploader_process_id == exited_process_id:
        os.kill(capturer_process_id, signal.SIGTERM)

    time.sleep(1)
    start_streaming_services()

def start_services():
    if not has_config_for_streaming():
        start_config_services()
    else:
        start_streaming_services()

if __name__ == '__main__':
    start_services()
```

ssdp.py

```
from ssdpy import SSDPServer
from config import PUBLIC_KEY_BARE_FILE_PATH

def run():
    with open(PUBLIC_KEY_BARE_FILE_PATH, 'r') as bare_public_file:
        public_key = bare_public_file.read()

    server = SSDPServer(public_key, device_type="ssdp:choicecam", port=1901)
    server.serve_forever()
```

videocapturer.py

```
import shlex
import os
import subprocess
from config import \
   SYMMETRIC_KEY_INFO_FILE_PATH, \
   VIDEO_OUTPUT_PATH, \
   PLAYLIST_FILE_PATH, \
   FONT_PATH
import time

def is_raspberry_pi():
   return os.uname().nodename == 'raspberrypi'

def get_unix_timestamp_with_timezone_offset():
   is_dst = time.daylight and time.localtime().tm_isdst > 0
   utc_offset = time.altzone if is_dst else time.timezone
   timestamp = round(time.time()) - utc_offset
   return timestamp

def construct_command():
   start_timestamp = get_unix_timestamp_with_timezone_offset()

   if is_raspberry_pi():
       command = 'raspivid -n -w 640 -h 360 -fps 25 -vf -t 86400000 -b 1800000 -ih -o - |
ffmpeg -y -i - '
       codec = 'h264_omx'
   else:
       command = 'ffmpeg -y -f avfoundation -framerate 30 -i "FaceTime:MacBook" '
       codec = 'libx264'

   command = command + \
         ' -b 500k' + \
         ' -vf scale=640:360' + \
         ' -c:v ' + codec + \
         ' -pix_fmt yuv420p' + \
         ' -preset superfast' + \
         ' -tune zerolatency' + \
         ' -vf "drawtext=fontfile=' + shlex.quote(FONT_PATH) + ': text=\'%{pts\:gmtime\:' +
\
        str(start_timestamp) + '\:%d-%m-%Y %T}\':' + \
        ' fontcolor=white: fontsize=h/10: x=(w-tw)/2: y=h-(2*lh):' + \
        ' box=1: boxcolor=0x00000000@1: boxborderw=5: alpha=0.5"' + \
        ' -f hls' + \
        ' -hls_time 3' + \
        ' -hls_list_size 1' + \
        ' -hls_start_number_source epoch ' + \
        ' -hls_flags temp_file' + \
```

```
         ' -hls_segment_filename ' + shlex.quote(os.path.join(VIDEO_OUTPUT_PATH,
"%012d.ts")) + \
         ' -hls_segment_type mpegts' + \
         ' -hls_key_info_file ' + shlex.quote(SYMMETRIC_KEY_INFO_FILE_PATH) + \
         ' ' + shlex.quote(PLAYLIST_FILE_PATH)
         # ' -hide_banner -loglevel panic' + \

   return command


def run():
   command = construct_command()
   process = subprocess.Popen(command, shell=True)
   process.wait()
```

videouploader.py

```
import shlex
import os
import subprocess
from crypto import encrypt_file
from config import \
   SYMMETRIC_KEY_FILE_PATH, \
   VIDEO_OUTPUT_PATH, \
   PLAYLIST_FILE_PATH, \
   provider_from_config
from urllib.request import urlopen
from urllib.error import HTTPError
import time

def load_segment_list_from_storage(provider):
   base_path = provider.base_path()
   segment_list = []

   try:
       segment_list_string = urlopen(base_path + 'segment_list').read().decode('utf-8')
       segment_list = segment_list_string.split("\n")
   except HTTPError as http_error:
       if http_error.code not in [404, 400, 403]:
           raise http_error

   return segment_list

def last_segment_from_segment_list(segment_list):
   if len(segment_list) == 0:
```

```
        return 0
    return int(max(segment_list))

def extract_thumbnail(video_file):
    segment_name = os.path.basename(video_file).replace('.ts', '')
    thumbnail_filename = os.path.join(VIDEO_OUTPUT_PATH, segment_name + '.jpg')
    unencrypted_thumbnail_filename = os.path.join(VIDEO_OUTPUT_PATH, segment_name +
'.unencrypted.jpg')
    playlist_filename = os.path.join(VIDEO_OUTPUT_PATH, segment_name + '.m3u8')

    playlist = """\
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:3
#EXT-X-MEDIA-SEQUENCE:1
#EXT-X-KEY:METHOD=AES-128,URI="{key_file}"
#EXTINF:3,
{segment_file}
#EXT-X-ENDLIST
    """.format(key_file=SYMMETRIC_KEY_FILE_PATH, segment_file=video_file)

    with open(playlist_filename, 'w') as playlist_file:
        playlist_file.write(playlist)

    if not os.path.isfile(unencrypted_thumbnail_filename):
        cmd = 'ffmpeg' + \
              ' -allowed_extensions ALL' + \
              ' -i ' + shlex.quote(playlist_filename) + \
              ' -vframes 1 -q:v 2 ' + unencrypted_thumbnail_filename

        process = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE)
        process.wait()

    if not os.path.isfile(thumbnail_filename):
        encrypt_file(SYMMETRIC_KEY_FILE_PATH, unencrypted_thumbnail_filename,
thumbnail_filename)
    os.remove(unencrypted_thumbnail_filename)
    os.remove(playlist_filename)
    return thumbnail_filename

def get_next_segment_file_to_sync(last_synced_segment_number):
    while True:
        next_unsynced_file = None
        for filename in sorted(os.listdir(VIDEO_OUTPUT_PATH)):
            if filename.endswith(".ts") and int(filename.replace('.ts', '')) >
last_synced_segment_number:
                next_unsynced_file = os.path.join(VIDEO_OUTPUT_PATH, filename)
```

```
                break

        if next_unsynced_file is None or not os.path.isfile(PLAYLIST_FILE_PATH):
            time.sleep(0.1)
            continue

        return next_unsynced_file


def watch_directory():
    provider = provider_from_config()
    segment_list = load_segment_list_from_storage(provider)
    last_synced_segment_number = last_segment_from_segment_list(segment_list)

    while True:
        segment_filename = get_next_segment_file_to_sync(last_synced_segment_number)
        thumbnail_filename = extract_thumbnail(segment_filename)

        provider.sync_file(key=os.path.basename(segment_filename),
                           path=segment_filename,
                           content_type='video/MP2T')

        provider.sync_file(key=os.path.basename(thumbnail_filename),
                           path=thumbnail_filename,
                           content_type='image/jpeg')

        segment_name = os.path.basename(segment_filename).replace('.ts', '')
        segment_list.append(segment_name)
        last_synced_segment_number = last_segment_from_segment_list(segment_list)

        provider.sync_string(key='segment_list',
                             content="\n".join(segment_list),
                             content_type='text/plain')

        os.remove(segment_filename)
        os.remove(thumbnail_filename)

def run():
    watch_directory()
```

webservice.py

```
import http.server
import socketserver
import traceback
from io import BytesIO
import base64
import json
import os
import sys
from Crypto.Cipher import PKCS1_v1_5
from providers import provider_factory
from Crypto.Signature import PKCS1_PSS
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from config import PRIVATE_KEY_FILE_PATH, save_config_for_streaming

PORT = 8000

class WebServiceHTTPRequestHandler(http.server.BaseHTTPRequestHandler):

   private_key = None

   def do_POST(self):
       try:
           content_length = int(self.headers['Content-Length'])
           body = self.rfile.read(content_length)

           message = self.decrypt_body(body)
           provider = provider_factory(message['provider_data'])

           if not provider.credentials_correct():
               self.send_signed_response(400, b"Incorrect credentials")
               return

           save_config_for_streaming(message)

           base_path = provider.base_path()
           self.send_signed_response(200, base_path.encode())

           # Configuration no longer needed, shutting down server.
           sys.exit(0)
       except Exception as e:
           print(e, file=sys.stderr)
           traceback.print_exc()
           self.send_signed_response(500, b"Unknown error")
```

```
    def decrypt_body(self, body):
        if self.private_key is None:
            self.init_private_key()

        encrypted_message = base64.b64decode(body)
        cipher = PKCS1_v1_5.new(self.private_key)
        decrypted_message = cipher.decrypt(encrypted_message, None)

        return json.loads(decrypted_message)

    def init_private_key(self):
        with open(PRIVATE_KEY_FILE_PATH, 'r') as file:
            private_key = file.read()

        self.private_key = RSA.importKey(private_key)

    def send_signed_response(self, status_code, body):
        if self.private_key is None:
            self.init_private_key()

        hash = SHA.new()
        hash.update(body)
        signer = PKCS1_PSS.new(self.private_key)
        signature = signer.sign(hash)

        self.send_response(status_code)
        self.send_header("Signature", base64.b64encode(signature).decode("utf-8"))
        self.end_headers()

        response = BytesIO()
        response.write(body)
        self.wfile.write(response.getvalue())

def run():
    if not os.path.isfile(PRIVATE_KEY_FILE_PATH):
        raise RuntimeError('Server private key not found under ' + PRIVATE_KEY_FILE_PATH)

    socketserver.TCPServer.allow_reuse_address = True
    with socketserver.TCPServer(("", PORT), WebServiceHTTPRequestHandler) as httpd:
        try:
            httpd.serve_forever()
        except:
            httpd.server_close()
            raise
```

9.4.2 Smartphone client

components/provider-forms/AwsProviderForm.js

```
import React from 'react';
import ProviderForm from './ProviderForm'

class AwsProviderForm extends ProviderForm {
   constructor(props) {
       super(props);
   }

   getType() {
       return 'aws';
   }

   getTitle() {
       return 'Amazon AWS S3';
   }

   getFields() {
       return [
           {'name': 'bucket_name', 'label': 'Bucket name', 'default': 'choicecam'},
           {'name': 'region', 'label': 'Region', 'default': 'eu-west-1'},
           {'name': 'key', 'label': 'Key', 'default': 'choicecam'},
           {'name': 'secret', 'label': 'Secret', 'default': 'choicecam'},
       ]
   }

   getBackgroundColor() {
       return '#ff9a00';
   }

   getLogo() {
       return require('../../images/aws.png');
   }
}

export default AwsProviderForm;
```

components/provider-forms/AzureProviderForm.js

```
import React from 'react';
import ProviderForm from './ProviderForm'

class AzureProviderForm extends ProviderForm {
   constructor(props) {
       super(props);
   }

   getType() {
       return 'azure';
   }

   getTitle() {
       return 'Microsoft Azure Blob Storage';
   }

   getFields() {
       return [
           {'name': 'container_name', 'label': 'Container name', 'default': 'choicecam'},
           {'name': 'connection_string', 'label': 'Connection string'},
           {'name': 'secret', 'label': 'Secret'},
       ]
   }

   getBackgroundColor() {
       return '#7eba01'
   }

   getLogo() {
       return require('../../images/azure.png');
   }
}

export default AzureProviderForm;
```

components/provider-forms/DigitalOceanProviderForm.js

```
import React from 'react';
import ProviderForm from './ProviderForm'

class DigitalOceanProviderForm extends ProviderForm {
   constructor(props) {
       super(props);
```

```
    }

    getType() {
        return 'digitalocean';
    }

    getTitle() {
        return 'Digital Ocean Spaces';
    }

    getFields() {
        return [
            {'name': 'bucketName', 'label': 'Bucket name'},
            {'name': 'region', 'label': 'Region', 'default': 'eu-west-1'},
            {'name': 'key', 'label': 'Key'},
            {'name': 'secret', 'label': 'Secret'},
        ]
    }

    getBackgroundColor() {
        return '#006aff'
    }

    getLogo() {
        return require('../../images/digitalocean.png');
    }
}

export default DigitalOceanProviderForm;
```

components/provider-forms/GoogleCloudProviderForm.js

```
import React from 'react';
import ProviderForm from './ProviderForm'

class GoogleCloudProviderForm extends ProviderForm {
    constructor(props) {
        super(props);
    }

    getType() {
        return 'googlecloud';
    }

    getTitle() {
```

```
        return 'Google Cloud Storage';
    }

    getFields() {
        return [
            {'name': 'bucketName', 'label': 'Bucket name'},
            {'name': 'key', 'label': 'Key'},
            {'name': 'secret', 'label': 'Secret'},
        ]
    }

    getBackgroundColor() {
        return '#4d8df5'
    }

    getLogo() {
        return require('../../images/googlecloud.png');
    }
}

export default GoogleCloudProviderForm;
```

components/provider-forms/ProviderForm.js

```
import React from 'react';
import {View, Text, StyleSheet, TouchableOpacity, Image} from 'react-native';
import Input from '../Input';
import Button from '../Button';
import { vw, vh, vmin, vmax } from 'react-native-expo-viewport-units';

class ProviderForm extends React.Component {

    constructor(props) {
        super(props);
        this.credentials = {};
        this.state = {
            expanded: false,
        };

        const fields = this.getFields();
        for (var i = 0; i < fields.length; i++) {
            var defaultValue = fields[i].default !== undefined ? fields[i].default : '';
            this.credentials[fields[i].name] = defaultValue;
        }
    }
```

```
    handleSave() {
        if (this.props.onSave) {
            this.props.onSave({
                type: this.getType(),
                credentials: this.credentials
            });
        }
    }

    onToggle() {
        this.setState({expanded: !this.state.expanded});
        console.log(this.state.expanded);
    }

    render() {
        const title = this.getTitle();
        const fields = this.getFields();
        const backgroundColor = this.getBackgroundColor();
        const logo = this.getLogo();

        var fieldElements = [];
        for (var i = 0; i < fields.length; i++) {
            var defaultValue = fields[i].default !== undefined ? fields[i].default : '';
            fieldElements.push(
                <Input
                    label={fields[i].label}
                    defaultValue={defaultValue}
                    onChangeText={function(name, component) {
                        return function(text) {
                            component.credentials[name] = text;
                        }
                    }(fields[i].name, this)}
                />
            );
        }

        return (
            <View style={[styles.container, {backgroundColor: backgroundColor}]}>
                <TouchableOpacity style={styles.titleButton}
onPress={this.onToggle.bind(this)}>
                    <Image style={styles.logo} source={logo} />
                    <Text style={styles.title}>{title}</Text>
                </TouchableOpacity>
                <View style={[styles.formView, this.state.expanded ? undefined :
styles.hidden]}>
                    {fieldElements}
```

```
                    <Button
                        text="Save"
                        onPress={this.handleSave.bind(this)}
                    />
                </View>
            </View>
        );
    }
}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        alignItems: 'center',
        justifyContent: 'center',
        borderRadius:10,
        borderWidth: 0,
        marginTop: vw(5),
        marginLeft: vw(5),
        marginRight: vw(5),
        padding: vw(5),
        shadowColor: "#000",
        shadowOffset: {
            width: 0,
            height: 2,
        },
        shadowOpacity: 0.25,
        shadowRadius: 3.84,
        elevation: 5,
    },
    title: {
        fontSize: 20,
        color: '#fff'
    },
    titleButton: {
        width: '100%',
        alignItems: 'center',
        justifyContent: 'center',
        flexDirection: 'row',
    },
    logo: {
        width: 35,
        height: 35,
        marginRight: 17,
    },
    hidden: {
        display: 'none',
```

```
    },
    formView: {
        flex: 1,
        alignItems: 'center',
        justifyContent: 'center',
        width: '100%',
    },
});

export default ProviderForm;
```

components/Aes128EncryptedImage.js

```
import React from 'react';
import {View, StyleSheet, Image, ActivityIndicator} from 'react-native';
import * as FileSystem from 'expo-file-system';
import { NativeModules, Platform } from 'react-native'
var Aes = NativeModules.Aes;
import base64 from 'react-native-base64'
import md5 from 'md5'

class Aes128EncryptedImage extends React.Component {

    constructor(props) {
        super(props);
        this.state = {
            key: props.symmetricKey,
            decryptedImagePath: null,
        };
    }

    componentWillReceiveProps(nextProps) {
        this.setState({
            key: nextProps.symmetricKey,
        });
        if (this.props.source != nextProps.source) {
            this.processSource(nextProps.source);
        }
    }

    processSource(source) {
        var uri = source['uri'];
        var uriHash = md5(uri);

        var extension = uri.split('.').pop();
```

```
        var localUri = FileSystem.cacheDirectory + uriHash + '.' + extension;
        var localUriDecoded = FileSystem.cacheDirectory + uriHash + '.decoded.' + extension;

        FileSystem.getInfoAsync(localUriDecoded).then(function (results) {
            if (results.exists) {
                this.setState({decryptedImagePath: localUriDecoded});
                return;
            }

            FileSystem.downloadAsync(
                uri,
                localUri
            )
                .then(function (result) {
                    return FileSystem.readAsStringAsync(result.uri, {
                        encoding: FileSystem.EncodingType.Base64
                    });
                }.bind(this))
                .then(function (contents) {
                    if (!this.state.key) {
                        throw new Error("Missing symmetric key.");
                    }
                    var iv = contents.substring(0, 48); // First 36 bytes in base64, chopped
to 32
                    iv = base64.decode(iv).substring(0, 32);
                    var cipherText = contents.substring(48);
                    return Aes.decrypt(cipherText, this.state.key, iv);
                }.bind(this))
                .then(function (decryptData) {
                    return FileSystem.writeAsStringAsync(localUriDecoded, decryptData, {
                        encoding: FileSystem.EncodingType.Base64
                    });
                }.bind(this))
                .then(function (contents) {
                    this.setState({decryptedImagePath: localUriDecoded});
                }.bind(this))
                .catch(function (error) {
                    Promise.all([
                        FileSystem.deleteAsync(localUriDecoded),
                        FileSystem.deleteAsync(localUri),
                    ]);

                    this.onError(error);
                }.bind(this));
        }.bind(this));
    }
```

```
    onError(error) {
        console.log(error);
        if (this.props.onError) {
            this.props.onError(error);
        }
    }

    render() {
        if (!this.state.decryptedImagePath) {
            return <View style={[{ alignItems: 'center', justifyContent: 'center' },
this.props.style]}>
                <ActivityIndicator size="large" />
            </View>;
        }

        var imageProps = this.props;
        delete imageProps['source'];
        delete imageProps['onError'];
        delete imageProps['key'];

        return <Image {...this.props} source={{uri: this.state.decryptedImagePath}}
onError={this.onError.bind(this)} />
    }
}


export default Aes128EncryptedImage;
```

components/Button.js

```
import React from 'react';
import { StyleSheet, TouchableOpacity, Text } from 'react-native';

class Button extends React.Component {
    static defaultProps = {
        theme: 'default',
    };

    render() {
        const { onPress, text } = this.props;

        return (
            <TouchableOpacity onPress={onPress} style={styles.container}>
                <Text style={styles.text}>{text}</Text>
            </TouchableOpacity>
```

```
        );
    }
}

const styles = StyleSheet.create({
    container: {
        backgroundColor: '#000000',
        marginVertical: 10,
        marginHorizontal: 15,
        borderRadius: 3,
        width: '100%'
    },
    text: {
        textAlign: 'center',
        color: 'rgba(255, 255, 255, 0.9)',
        fontWeight: '600',
        fontSize: 16,
        padding: 15,
    },
});

export default Button;
```

components/Input.js

```
import React from 'react';
import {
    Text,
    View,
    TextInput,
    StyleSheet,
} from 'react-native';

class Input extends React.Component {
    render() {
        const { label, ...props } = this.props;
        return (
            <React.Fragment>
                <Text style={styles.label}>{label}</Text>
                <View style={styles.inputContainer}>
                    <TextInput
                        style={styles.input}
                        {...props}
                    />
                </View>
```

```
            </React.Fragment>
        );
    }
}

const styles = StyleSheet.create({
    label: {
        color: 'rgba(255,255,255,0.5)',
        paddingHorizontal: 20,
        marginBottom: 5,
        marginTop: 10,
        fontSize: 16,
    },
    inputContainer: {
        backgroundColor: '#fff',
        paddingHorizontal: 20,
        width: '100%',
        borderRadius: 3,
    },
    input: {
        color: '#000',
        fontSize: 16,
        paddingVertical: 10,
    },
});

export default Input;
```

screens/CameraMonitor.js

```
import * as React from 'react';
import {FlatList, StyleSheet, Text, View, TouchableOpacity, ActivityIndicator} from
'react-native';
import Aes128EncryptedImage from '../components/Aes128EncryptedImage';
import {Video} from 'expo-av';
import Moment from 'moment';
import { vw, vh, vmin, vmax } from 'react-native-expo-viewport-units';
import VideoServer from "../VideoServer";
import Button from "../components/Button";

class CameraMonitor extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            key: props.navigation.state.params.symmetricKey,
```

```
            playlistUri: null,
            thumbnailList: [],
            isLive: null,
            isLoading: true,
            videoIsLoading: true,
        };

        this.video = null;
        this.videoServer = new VideoServer(
            props.navigation.state.params.symmetricKey,
            props.navigation.state.params.basePath
        );

        this.videoServer.onUrlChange(this.setVideoSource.bind(this));
        this.videoServer.onThumbnailListChange(this.setThumbnailList.bind(this));
        this.videoServer.onLiveStatusChange(this.setLiveStatus.bind(this));
        this.videoServer.start();
    }

    handleVideoRef(ref) {
        this.video = ref;
    }

    setVideoSource(uri) {
        if (this.video) {
            this.video.stopAsync().then(function() {
                this.setState({playlistUri: uri});
            }.bind(this));
        } else {
            this.setState({playlistUri: uri});
        }
    }

    setThumbnailList(list) {
        if (list.length > 0 && this.state.isLoading) {
            this.setState({isLoading: false});
        }
        this.setState({thumbnailList: list.reverse()});
    }

    setLiveStatus(isLive) {
        this.setState({isLive: isLive});
    }

    onListItemPress(item) {
        this.videoServer.seekTo(item.sequence);
    }
```

```
   onGoLiveButtonPress() {
       this.videoServer.goLive();
   }

   onPlaybackStatusUpdate(playbackStatus) {
       this.setState({
           videoIsLoading: !playbackStatus.isPlaying
       });
   }

   render() {
       if (this.state.isLoading) {
           return <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
               <ActivityIndicator size="large" />
               <Text style={styles.loadingText}>Waiting for video feed...</Text>
           </View>
       }
       Moment.locale('en');
       return (
           <View style={styles.container}>
               <View style={[styles.videoLoading, this.state.videoIsLoading === true ? {} :
{display: 'none'}]}>
                   <ActivityIndicator size="large" />
               </View>
               <Video
                   ref={this.handleVideoRef.bind(this)}
                   source={{uri: this.state.playlistUri}}
                   shouldPlay
                   useNativeControls={false}
                   style={[styles.video, this.state.videoIsLoading === true ? {display:
'none'} : {}]}
                   onPlaybackStatusUpdate={this.onPlaybackStatusUpdate.bind(this)}
               />
               <View style={[styles.liveIndicator, this.state.isLive === true ?
styles.liveIndicatorLive : {}]}>
                   <Text style={styles.liveIndicatorText}>
                       {this.state.isLive === true ? 'Live' : ''}
                       {this.state.isLive === false ? 'Watching Recording' : ''}
                       {this.state.isLive === null ? 'Loading...' : ''}
                   </Text>
                   <TouchableOpacity style={styles.goLiveButton}
onPress={this.onGoLiveButtonPress.bind(this)}>
                       <Text style={[
                               styles.goLiveButtonText,
                               this.state.isLive !== false ? {display: 'none'} : {}
                           ]}>Go Live</Text>
```

```
                    </TouchableOpacity>
                </View>
                <FlatList
                    style={styles.list}
                    data={this.state.thumbnailList}
                    renderItem={({item, index}) =>
                        <TouchableOpacity onPress={ () => this.onListItemPress(item)}>
                            <View style={[styles.listItem, index === 0 ? {display: 'none'} :
{}]}>
                                <Aes128EncryptedImage
                                    style={styles.thumbnail}
                                    source={{uri: item.thumbnailFile}}
                                    symmetricKey={this.state.key}/>
                            </View>
                        </TouchableOpacity>
                    }
                    keyExtractor={(item, index) => item.sequence}
                />
            </View>
        );
    }
}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        backgroundColor: '#fff',
        alignItems: 'center',
        justifyContent: 'center',
    },
    video: {
        width: vw(100),
        height: vw(56.25),
    },
    videoLoading: {
        width: vw(100),
        height: vw(56.25),
        alignItems: 'center',
        justifyContent: 'center',
    },
    listItem: {
        flex: 1,
        flexDirection: 'row'
    },
    list: {
        width: '100%'
    },
```

```
   thumbnail: {
       width: vw(100),
       height: vw(56.25),
   },
   liveIndicator: {
       backgroundColor: '#000000',
       width: '100%',
       justifyContent: 'center',
       alignItems: 'center',
       flexDirection: 'row',
   },
   liveIndicatorLive: {
       backgroundColor: '#008800',
   },
   liveIndicatorText: {
       textAlign: 'center',
       color: 'white',
       fontSize: 16,
       padding: 15,
   },
   goLiveButton: {
       backgroundColor: 'blue',
       borderRadius: 3,
   },
   goLiveButtonText: {
       fontSize: 16,
       padding: 10,
       color: 'white',
       fontWeight: '600',
   },
   loadingText: {
       fontSize: 20,
       marginTop: vw(10),
       marginBottom: vw(10),
       textAlign: 'center'
   },
});

export default CameraMonitor;
```

screens/CodeScanner.js

```
import React, { useState, useEffect } from 'react';
import { Text, View, StyleSheet, Button } from 'react-native';
import { BarCodeScanner } from 'expo-barcode-scanner';

class CodeScanner extends React.Component {

   constructor(props) {
       super(props);
       this.state = {
           hasPermission: null,
           scanned: false,
       };
   }

   componentDidMount() {
       (async () => {
           const { status } = await BarCodeScanner.requestPermissionsAsync();
           this.setState({ hasPermission: status === 'granted' });
       })();
   }

   handleBarCodeScanned ({ type, data }) {
       function validatePublicKeyBase64 (data) {
           var base64regex =
/^([0-9a-zA-Z+/]{4})*(([0-9a-zA-Z+/]{2}==)|([0-9a-zA-Z+/]{3}=))?$/;

           if (!base64regex.test(data)) {
               return false;
           }
           if (data.length !== 392) {
               return false;
           }
           return true;
       }

       if (!validatePublicKeyBase64(data)) {
           console.log('Invalid QR code: ' + data);
           return;
       }

       this.setState({ scanned: true });

       this.props.navigation.navigate('DeviceSearch', {
           publicKey: data,
```

```
        });
    }

    render() {

        if (this.state.hasPermission === null) {
            return <Text>Requesting for camera permission</Text>;
        }
        if (this.state.hasPermission === false) {
            return <Text>No access to camera</Text>;
        }


        return (
            <View
                style={{
                    flex: 1,
                    flexDirection: 'column',
                    justifyContent: 'flex-end',
                }}>
                <BarCodeScanner
                    onBarCodeScanned={this.state.scanned ? undefined :
this.handleBarCodeScanned.bind(this)}
                    style={StyleSheet.absoluteFillObject}
                    barCodeTypes={[BarCodeScanner.Constants.BarCodeType.qr]}
                />
            </View>
        );
    }
}

export default CodeScanner;
```

screens/DeviceSearch.js

```
import React from 'react';
import {ActivityIndicator, View, Text, StyleSheet} from 'react-native';
import dgram from 'react-native-udp';
import base64 from 'react-native-base64';
import {vw} from "react-native-expo-viewport-units";

class DeviceSearch extends React.Component {

    constructor(props) {
```

```
        super(props);
        this.socket = null;
        this.searchInterval = null;
    }

    componentDidMount() {
        this.searchDevice();
    }

    componentDidUpdate() {
        this.searchDevice();
    }

    componentWillUnmount() {
        this.stopSearch();
    }

    searchDevice() {
        this.socket = dgram.createSocket('udp4');
        var publicKeyToSearch = this.props.navigation.state.params.publicKey;

        function byteArrayToString(array) {
            var result = "";
            for (var i = 0; i < array.length; i++) {
                result += String.fromCharCode(array[i]);
            }
            return result;
        }

        function parseHeader(message) {
            const lines = message.split("\r\n");
            const firstLine = lines.shift();

            if ("NOTIFY * HTTP/1.1" !== firstLine) {
                return null;
            }

            var headers = {};

            for (var i = 0; i < lines.length; i++) {
                const parts = lines[i].split(":", 2);
                if (parts.length !== 2) {
                    continue;
                }

                headers[parts[0].toLowerCase().trim()] = parts[1].trim();
            }
```

```
            return headers;
        }

        function emitSearch() {
            var message = "M-SEARCH * HTTP/1.1\r\n" +
                "HOST: 239.255.255.250:1901\r\n" +
                "MAN: \"ssdp:discover\"\r\n" +
                "MX: 1\r\n" +
                "ST: ssdp:choicecam";

            this.socket.send(base64.encode(message), 0, message.length, 1901,
'239.255.255.250', function(err) {
                if (err) throw err;
            });
        }

        this.socket.on('message', function(messageBytes, serverInfo) {
            const message = byteArrayToString(messageBytes);
            const headers = parseHeader(message);
            if (!headers || !headers['usn']) {
                return;
            }

            if (headers['usn'] !== publicKeyToSearch) {
                return;
            }

            this.stopSearch();
            this.props.navigation.navigate('DeviceSetup', {
                publicKey: publicKeyToSearch,
                ipAddress: serverInfo.address,
            });

        }.bind(this));

        this.searchInterval = window.setInterval(emitSearch.bind(this), 1000);
        emitSearch.call(this);
    }

    stopSearch() {
        if (this.searchInterval) {
            clearInterval(this.searchInterval);
            this.searchInterval = null;
        }
        if (this.socket) {
            this.socket.close();
```

```
            this.socket = null;
        }
    }

    render() {
        const { params } = this.props.navigation.state;

        return (
            <View style={styles.container}>
                <ActivityIndicator size="large" />
                <Text style={styles.title}>Searching for camera...</Text>
                <Text style={styles.text}>Please make sure the camera is on the same
network.</Text>
                <Text style={styles.publicKey}>Public key: {params.publicKey}</Text>
            </View>
        );
    }
}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        alignItems: 'center',
        justifyContent: 'center',
        margin: vw(10)
    },
    title: {
        fontSize: 20,
        marginTop: vw(10),
        marginBottom: vw(10),
        textAlign: 'center'
    },
    text: {
        textAlign: 'center',
        marginBottom: vw(10),
    },
    publicKey: {
        fontSize: 8,
        textAlign: 'center',
        color: '#999'
    },
});

export default DeviceSearch;
```

screens/DeviceSetup.js

```
import React from 'react';
import {ActivityIndicator, ScrollView, StyleSheet, View} from 'react-native';
import { writeConfig } from '../Configuration';
import { vw, vh, vmin, vmax } from 'react-native-expo-viewport-units';
import { RSA, RSAKeychain } from 'react-native-rsa-native';
import { NativeModules } from 'react-native'
var Aes = NativeModules.Aes;

import AwsProviderForm from '../components/provider-forms/AwsProviderForm';
import DigitalOceanProviderForm from
'../components/provider-forms/DigitalOceanProviderForm';
import GoogleCloudProviderForm from '../components/provider-forms/GoogleCloudProviderForm';
import AzureProviderForm from '../components/provider-forms/AzureProviderForm';

class DeviceSetup extends React.Component {

   constructor(props) {
       super(props);
       this.state = {
           loading: false,
       }
   }

   handleSave(providerData) {
       function verifyResponse(response, publicKey) {
           return new Promise(function(resolve, reject) {
               const headers = response.headers.entries();

               for (var pair of headers) {
                   if (pair[0] !== 'signature') {
                       continue;
                   }
                   var signature = pair[1];
                   break;
               }

               if (!signature) {
                   reject();
                   return;
               }

               response.text().then(function (responseText) {
                   RSAKeychain.verify(signature, responseText, publicKey)
                       .then(function (verified) {
```

```
                    if (verified) {
                        resolve({
                            response: response,
                            responseText: responseText
                        });
                        return;
                    }
                    reject();
                }).catch(function () {
                reject();
            });
        });
    });
}

function saveBasePathAndKey(basePath, symmetricKey) {
    Promise.all([
        writeConfig('base_path', basePath),
        writeConfig('symmetric_key', symmetricKey),
    ]).then((values) => {
        this.props.navigation.navigate('StartupRouter');
    });
}

function showError(message) {
    alert(message);
    this.setState({
        loading: false,
    });
}

this.setState({
    loading: true,
});

var component = this;
var publicKey = this.props.navigation.state.params.publicKey;
var ipAddress = this.props.navigation.state.params.ipAddress;

Aes.randomKey(16).then(function (symmetricKey) {
    var message = {
        'symmetric_key': symmetricKey,
        'provider_data': providerData
    };

    message = JSON.stringify(message);
    publicKey = "-----BEGIN PUBLIC KEY-----" + publicKey + "-----END PUBLIC
```

```
KEY-----";

            RSA.encrypt(message, publicKey)
                .then(encodedMessage => {
                    return fetch('http://' + ipAddress + ':8000/', {
                        method: 'POST',
                        body: encodedMessage
                    });
                }).catch(function () {
                    throw new Error("Unexpected error happened. Please try again.");
                })
                .then(function (response) {
                    return verifyResponse(response, publicKey)
                }).catch(function () {
                    throw new Error("The identity of the webServer could not be verified.");
                }).then(function (result) {
                    var response = result.response;
                    var responseText = result.responseText;
                    if (!response.ok) {
                        throw new Error(responseText);
                    }

                    saveBasePathAndKey.call(component, responseText, symmetricKey);
                }).catch(function(error) {
                    showError.call(component, error.message);
                });
        });
    }

    render() {
        return (
            <View style={ styles.container }>
                <ScrollView style={ styles.providerList }>
                    <AwsProviderForm onSave={this.handleSave.bind(this)} />
                    <DigitalOceanProviderForm onSave={this.handleSave.bind(this)} />
                    <GoogleCloudProviderForm onSave={this.handleSave.bind(this)} />
                    <AzureProviderForm onSave={this.handleSave.bind(this)} />

                    <View style={ styles.listBottomPad } />
                </ScrollView>
                <View style={[styles.loadingScreen, this.state.loading ? {} : {display:
'none'}]}>
                    <ActivityIndicator size="large" />
                </View>
            </View>
        );
    }
```

```
}

const styles = StyleSheet.create({
   container: {
       flex: 1,
   },
   providerList: {
       flex: 1,
   },
   listBottomPad: {
       height: vh(50),
   },
   listItem: {
       flex: 1,
       flexDirection: 'row'
   },
   loadingScreen: {
       top: 0,
       left: 0,
       right: 0,
       bottom: 0,
       alignItems: 'center',
       justifyContent: 'center',
       position: 'absolute',
       backgroundColor: '#fff',
   }
});


export default DeviceSetup;
```

screens/StartupRouter.js

```
import React from 'react';
import { ActivityIndicator, View } from 'react-native';
import {readConfig, deleteConfig} from '../Configuration'

class StartupRouter extends React.Component {

   static firstRun = true;

   componentDidMount() {
       if (StartupRouter.firstRun) {
           Promise.all([
               deleteConfig('base_path'),
```

```
                deleteConfig('symmetric_key'),
            ]).then(function() {
                this.routeBasedOnConfig();
                StartupRouter.firstRun = false;
            }.bind(this));
        } else {
            this.routeBasedOnConfig();
        }
    }

    routeBasedOnConfig() {
        const navigation = this.props.navigation;

        Promise.all([
            readConfig('base_path'),
            readConfig('symmetric_key'),
        ]).then((values) => {
            var basePath = values[0];
            var symmetricKey = values[1];

            if (symmetricKey && basePath) {
                navigation.navigate('Configured', {
                    symmetricKey: symmetricKey,
                    basePath: basePath,
                });
                return;
            }

            navigation.navigate('Unconfigured');
        });
    }

    render() {
        return (
            <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
                <ActivityIndicator size="large" />
            </View>
        );
    }
}

export default StartupRouter;
```

App.js

```
import React from 'react';
import Navigation from './Navigation';

console.disableYellowBox = true;

class App extends React.Component {
   render() {
       return (
           <Navigation />
       );
   }
}

export default App;
```

Configuration.js

```
import * as SecureStore from 'expo-secure-store';

export async function readConfig(key) {
   return await SecureStore.getItemAsync(key);
}

export async function writeConfig(key, value) {
   await SecureStore.setItemAsync(
       key,
       value,
       {
           keychainAccessible: SecureStore.AFTER_FIRST_UNLOCK
       }
   );
}

export async function deleteConfig(key) {
   return await SecureStore.deleteItemAsync(key);
}
```

Navigation.js

```
import React from 'react';
import {
   createSwitchNavigator,
   createAppContainer,
} from 'react-navigation';

import { createStackNavigator } from 'react-navigation-stack';

import CameraMonitor from './screens/CameraMonitor';
import CodeScanner from './screens/CodeScanner';
import StartupRouter from './screens/StartupRouter';
import DeviceSearch from './screens/DeviceSearch';
import DeviceSetup from "./screens/DeviceSetup";

const ConfigureStack = createStackNavigator({
   CodeScanner: {
       screen: CodeScanner,
       navigationOptions: {
           headerTitle: 'Scan Camera QR Code',
       },
   },
   DeviceSearch: {
       screen: DeviceSearch,
       navigationOptions: {
           headerShown: false,
       },
   },
   DeviceSetup: {
       screen: DeviceSetup,
       navigationOptions: {
           headerTitle: 'Select your storage provider',
       },
   },
});

const AppNavigator = createAppContainer(createSwitchNavigator({
   StartupRouter: {
       screen: StartupRouter,
   },
   Unconfigured: {
       screen: ConfigureStack,
   },
   Configured: {
       screen: CameraMonitor,
```

```
   },
}));

export default AppNavigator;
```

Utilities.js

```
export function hexToBase64(hex) {
   var characters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
   var base64Codes = [];

   function appendCodeForSixBitCode(sixBitCode) {
       base64Codes.push(characters.substring(sixBitCode, sixBitCode + 1));
   }
   function appendPadding() {
       base64Codes.push('=');
   }

   var i = 0;
   var previousByte;
   var remainingBits = 0;
   while (true) {
       var byte = parseInt(hex.substring(i, i + 2), 16);
       switch (remainingBits) {
           case 0:
               appendCodeForSixBitCode(byte >>> 2);
               remainingBits = 2;
               break;
           case 2:
               appendCodeForSixBitCode(((previousByte & 3) << 4) |  (byte >>> 4));
               remainingBits = 4;
               break;
           case 4:
               appendCodeForSixBitCode(((previousByte & 15) << 2) |  (byte >>> 6));
               appendCodeForSixBitCode(byte & 63);
               remainingBits = 0;
       }


       previousByte = byte;
       i = i + 2;
       if (i >= hex.length) {
           switch (remainingBits) {
               case 2:
```

```
                    appendCodeForSixBitCode((byte & 3) << 4);
                    appendPadding();
                    appendPadding();
                    break;
                case 4:
                    appendCodeForSixBitCode((byte & 15) << 2);
                    appendPadding();
                    break;
            }
            break;
        }
    }

    return base64Codes.join("");
}
```

VideoServer.js

```
import StaticServer from "react-native-static-server";
import * as FileSystem from "expo-file-system";
import {hexToBase64} from './Utilities'

export default class VideoServer
{
    PORT = 7777;

    constructor(key, basePath) {
        this.key = key;
        this.basePath = basePath;
        this.playlistFilename = null;
        this.segments = [];
        this.currentSequence = null;
        this.onUrlChangeCallback = function() {};
        this.onThumbnailListChangeCallback = function() {};
        this.onLiveStatusChangeCallback = function() {};
    }

    start() {
        this.webServer = this.startWebServer();

        this.newPlaylistFileName();
        this.refreshSegmentList()
            .then(function () {
                return this.rewritePlaylistFile();
```

```
            }.bind(this)).then(function () {
                this.onUrlChangeCallback(this.currentPlaylistUri());
                this.onThumbnailListChangeCallback(this.segments.slice());
                this.onLiveStatusChangeCallback(true);
            }.bind(this))
            .catch(function (error) {
                console.log(error);
            });

        setInterval(function() {
            this.refreshSegmentList()
                .then(function () {
                    return this.rewritePlaylistFile();
                }.bind(this)).then(function () {
                    this.onThumbnailListChangeCallback(this.segments.slice());
                }.bind(this))
        }.bind(this), 1000);
    }

    seekTo(sequenceNumber) {
        this.newPlaylistFileName();
        this.currentSequence = sequenceNumber;
        this.rewritePlaylistFile(true)
            .then(function () {
                this.onUrlChangeCallback(this.currentPlaylistUri());
                this.onLiveStatusChangeCallback(false);
            }.bind(this));
    }

    goLive() {
        this.newPlaylistFileName();
        this.currentSequence = null;
        this.rewritePlaylistFile(true)
            .then(function () {
                this.onUrlChangeCallback(this.currentPlaylistUri());
                this.onLiveStatusChangeCallback(true);
            }.bind(this));
    }

    onUrlChange(callback) {
        this.onUrlChangeCallback = callback;
    }

    onThumbnailListChange(callback) {
        this.onThumbnailListChangeCallback = callback;
    }
```

```
onLiveStatusChange(callback) {
    this.onLiveStatusChangeCallback = callback;
}

newPlaylistFileName() {
    this.playlistFilename = 'playlist' + Math.random() + '.m3u8';
}

currentPlaylistUri() {
    return 'http://localhost:' + this.PORT + '/' + this.playlistFilename;
}

startWebServer() {
    var documentRoot = FileSystem.documentDirectory.replace('file://', '');
    this.webServer = new StaticServer(this.PORT, documentRoot, {localOnly : true});
    this.webServer.start();
}

refreshSegmentList()
{
    return new Promise(function (resolve, reject) {
        fetch(this.basePath + 'segment_list?cachebust=' + Math.random())
            .then(response => response.text())
            .then((response) => {
                var segmentList = response;
                if (segmentList === "") {
                    return;
                }

                var segmentNames = segmentList.split("\n");
                var segments = [];
                for (var i = 0; i < segmentNames.length; i++) {
                    var segmentName = segmentNames[i];
                    var item = {
                        'videoFile': this.basePath + segmentName + '.ts',
                        'thumbnailFile': this.basePath + segmentName + '.jpg',
                        'sequence': i + 1,
                    };

                    segments.push(item);
                }

                this.segments = segments;
                resolve();
            })
            .catch(reject);
    }.bind(this));
```

```
    }

    rewritePlaylistFile() {
        const keyInBase64 = hexToBase64(this.key);
        var playlistStartSequence;
        if (null !== this.currentSequence) {
            playlistStartSequence = this.currentSequence;
        } else {
            var tailSizeForLiveVideo = 4;
            playlistStartSequence = Math.max(1, this.segments.length - tailSizeForLiveVideo +
1);
        }

        var playlistFile = "#EXTM3U\n" +
            (null !== this.currentSequence ? "#EXT-X-PLAYLIST-TYPE:EVENT\n" : "") +
            "#EXT-X-VERSION:3\n" +
            "#EXT-X-TARGETDURATION:8\n" +
            "#EXT-X-ALLOW-CACHE:NO\n" +
            "#EXT-X-MEDIA-SEQUENCE:" + playlistStartSequence + "\n" +
            "#EXT-X-KEY:METHOD=AES-128,URI=\"data:text/plain;base64," + keyInBase64 + "\"\n";

        for (var i = 0; i < this.segments.length; i++) {
            var segment = this.segments[i];

            if (segment.sequence < playlistStartSequence) {
                continue;
            }
            playlistFile = playlistFile + "#EXTINF:8.333333,\n" +
                this.segments[i].videoFile + "?cachebust=" + this.playlistFilename + "\n";
        }

        var uri = FileSystem.documentDirectory + this.playlistFilename;
        return FileSystem.writeAsStringAsync(uri, playlistFile);
    }


}
```

9.5 List of software dependencies

9.5.1 Camera server

Library name	Version
azure-core	1.5.0
azure-storage-blob	12.3.1
boto3	1.13.6
botocore	1.16.6
cachetools	4.1.0
certifi	2020.4.5.1
cffi	1.14.0
chardet	3.0.4
colorlog	4.1.0
cryptography	2.9.2
docutils	0.15.2
idna	2.9
isodate	0.6.0
jmespath	0.9.5
miniupnpc	2.0.2
msrest	0.6.14
Naked	0.1.31
oauthlib	3.1.0
pathtools	0.1.2
Pillow	7.1.2
protobuf	3.12.2
psutil	5.7.0
pyasn1	0.4.8
pyasn1-modules	0.2.8
pycparser	2.2

pycrypto	2.6.1
python-dateutil	2.8.1
pytz	2020.1
PyYAML	5.3.1
qrcode	6.1
requests	2.23.0
requests-oauthlib	1.3.0
rsa	4
s3transfer	0.3.3
shellescape	3.8.1
six	1.14.0
ssdpy	0.2.4
urllib3	1.25.9

9.5.2 Smartphone client

Library name	**Version**
@react-native-community/masked-view	0.1.10
expo	37.0.3
expo-av	8.1.0
expo-barcode-scanner	8.1.0
expo-file-system	8.1.0
expo-secure-store	8.1.0
expo-splash-screen	0.2.3
expo-updates	0.2.0
md5	2.2.1

moment	2.25.3
react	16.9.0
react-dom	16.9.0
react-native	0.61.5
react-native-aes-crypto	https://github.com/tcz/react-native-aes.git
react-native-base64	0.0.2
react-native-expo-viewport-units	0.0.8
react-native-gesture-handler	1.6.0
react-native-reanimated	1.7.0
react-native-rsa-native	2.0.0
react-native-safe-area-context	1.0.2
react-native-screens	2.7.0
react-native-static-server	0.4.2
react-native-udp	3.1.0
react-native-unimodules	0.9.0
react-native-web	0.11.7
react-navigation	4.3.9
react-navigation-stack	2.5.1

10 Self-evaluation

Overall I'm happy with the project.

The prototype, while admittedly not perfect, turned out to be decent in terms of its design and code quality. I wish I had had more time to round the rough edges, add tests, etc. Neither Python nor React Native are my usual platforms in terms of software development, and I'm glad I left my comfort zone for this project. The camera is not yet ready to replace the NestCam I own, but with a little more work after finishing the academic year it will be and I'll be happy to unplug that little spy device in my living room. It will complement well my homemade rotary phone voice assistant [75].

In terms of the report itself, I'm a little worried that I failed to convey the idea and how it would be applicable in a broader sense. Perhaps I spent too much time going into details irrelevant to the core of the project, like HLS video streaming, SSDP and the like. Yet I thought it was important that I demonstrate my knowledge I acquired during my studies even if it's not instrumental to understanding the main theme of the project.