Towards more privacy-preserving and practical Internet-of-Things Devices

A Thesis
submitted to the Faculty of the
Graduate School of Arts and Sciences
of Georgetown University
in partial fulfillment of the requirements for the
degree of
Master of Science
in Computer Science

By

Feiyang Yu, B.S.

Washington, DC
April 19, 2021

Towards more privacy-preserving and practical Internet-of-Things Devices

Feiyang Yu, B.S.

Thesis Advisor: Micah Sherr

Abstract

IoT (Internet-of-Things) devices have seen widespread deployment over the past decade. Since they enable home appliances to connect to the Internet, IoT services have made daily life much more convenient. However, on the other hand, they have also made home appliances open to the public on the Internet and thus have introduced vulnerabilities and threats to user privacy that were previously not possible on disconnected devices. Security and privacy incidents of IoT devices are reported every year, and the topic has received considerable attention from both security researchers and IoT manufacturers.

This thesis first surveys the background of IoT devices, including what is provided on the market and the feature sets of existing products. We then look at existing studies which discuss the security and privacy threats of IoT systems.

The main contribution of this thesis is a privacy-preserving framework for generalized IoT devices. The framework is designed to defend against the vulnerabilities and threats mentioned above. It offers security and privacy while supporting the features offered by modern IoT devices and is suitable for typical home networks. We also propose communication and attacker models in this scenario and access the framework's security and usability features.

Finally, we conduct a case study, which focuses on a specific type of IoT system, namely Internet-enabled video doorbells. We implement a privacy-preserving doorbell

system using the proposed design, evaluate its usability and compare it with existing products.

INDEX WORDS: IoT, privacy

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Background on IoT devices

Nowadays, thanks to the rapid growth of material science, it has been possible to put microprocessors into various sizes of home appliances. The microprocessors enable home appliances to connect to the Internet and communicate with other devices. Such development has brought human society into the era of Internet-of-Things (IoT).

Unlike traditional computing platforms, IoT devices are different in 1) operating continuously, 2) receiving data from sensors, 3) constantly communicating with other devices to report sensitive data. Because of the nature of IoT, the devices are usually located publicly on the Internet.

There is also a tremendous business potential for IoT devices. There have been a significant number of IoT devices on the market, including healthcare devices (monitors, etc.), remote controllers (TV, oven, recorder, etc.), communication devices (phones, VoIP, etc.), and many more [13]. These IoT devices usually have the following common features:

- Receiving data from the sensor.
- Processing data.
- Sending sensitive data to end-user devices.

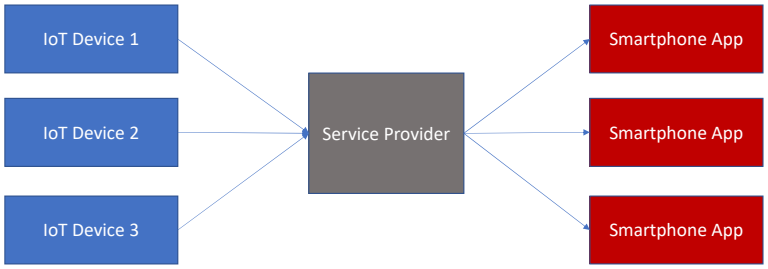

Figure 1.1: A typical centralized structure for IoT systems

Typically, IoT systems adopts the structure shown in *Fig. 1.1*. In such a structure, multiple IoT devices connect to some centralized servers. IoT devices send alerts to these servers, and the servers are responsible for distributing messages to end-user devices (usually smartphone apps).

1.2 Motivation: the need for a privacy-preserving framework

However, connecting home appliances to the Internet introduces vulnerabilities to malicious attacks. An attacker may obtain private information, such as the user's geographic info, preference, or payment information. Attackers may even take over

the control of such appliances and use them for further malicious attacks into other systems in the home network.

1.2.1 Privacy threats of IoT devices

In reality, there have been many severe security and privacy incidents with commercial IoT devices.

In 2016, a large DDoS attack was launched on service provider Dyn using the Mirai Botnet [3], which brought down sites including Twitter, the Guardian, Netflix and many more [32]. Devices, including digital cameras and DVR players, were attacked and infected with malware. In early 2017, the FDA confirmed that St. Jude Medical's implantable cardiac devices have vulnerabilities that allow hackers to read the data and even take over the device [17]. In late 2019, the Ring doorbell had suffered from a data leak, exposing essential user data, including emails, passwords, time zones, and names given to specific devices [22].

There already exists a rich literature that examines the security and privacy properties of these devices [1] [12] [25] [4]. As we describe in the next chapter, IoT systems' security and privacy properties of have been studied widely.

1.2.2 Challenges

There are a few challenges for privacy-preserving settings on home IoT devices.

- Push notifications. While being an essential part of IoT services, push notifications are the first obstacle to building a privacy-preserving framework. Ideally, a privacy-preserving system would like to avoid any point of centralization, and so does the push notification part. However, it is hard to keep all valuable functions while achieving complete decentralization on modern mobile systems. As

we described later in the discussion section, one would have to either adapt an existing commercial service or implement her own system with fewer data and energy efficiency.

- Limited computational power. Home IoT devices are usually built in small-size and do not have high computational power as traditional computing platforms such as desktop computers and cloud servers. The limited power is another difficulty building privacy-preserving systems, as the IoT devices will not be able to do heavy computations on-device.

1.3 Overview of approach

In this thesis, we take a different track, and rather than attempt to examine one aspect of IoT devices up close, we adopt a more holistic approach. We ask the question, "is it possible to create a privacy-preserving IoT device that offers the same features as commercially-available IoT devices?" Our goal is to determine the feasibility and potential performance costs of duplicating the capabilities of feature-rich IoT devices but in a privacy-preserving way.

In our approach, we use Tor (The Onion Router) for the purpose of obscuring the communication between IoT devices and end-user devices. Tor has a few excellent features for the task, which we describe later in this chapter.

1.4 Scientific questions and contributions

In answer to the above questions and challenges, this Master's thesis proposes a novel privacy-preserving framework for Internet-of-Things (IoT) devices that obscures the devices' traffic patterns and implements and an example application that uses this

framework. A core goal of this work is to develop feature-rich IoT devices that offer similar features to existing (but non-privacy-preserving) IoT devices.

Besides, we conduct a case study that considers a specific type of home IoT device, namely an Internet-enabled video doorbell. The specific system includes a number of interesting features that may be challenging in privacy-preserving settings, which include but not limited to face and motion detection, video and audio communications and push notifications. We tackle the challenges and implement a feasible system based on our proposed design.

Chapter 2

Related work

In this chapter, we present prior research related to IoT security. We present the previous approaches and discuss the differences between our approach and existing work.

In addition to security, we also review previous research related to IoT *anonymity*. Several papers attempt to use the Tor network in concert with IoT systems to achieve anonymity, but the problem has not been studied in great depth, as we discuss below.

2.1 IoT security and privacy

IoT security and privacy has been studied widely. Alrawi *et al.* [2] provide an overview of security study challenges in modern IoT systems. In their work, they propose a modeling methodology to study home-based IoT devices and evaluate their security posture based on component analysis. Lin *et al.* [18] survey existing solutions for enhancing IoT security and identify key feature requirements for trusted smart home systems. They point out two key technologies, support for system auto-configuration and automatic update of system firmware.

In terms of privacy-preserving structures for IoT systems, researchers have proposed many protocols covering different aspects of privacy. Song *et al.* [26] design a privacy-preserving communication protocol for IoT applications. Their protocol enables IoT appliances and sensors to communicate with a central controller securely

and ensures data integrity and authentication by incorporating Message Authentication Codes (MACs, which protects messages' integrity and authenticity by allowing verifiers to detect any changes to the message content) to the data transmission. While we are not focused on protecting the veracity of IoT measurements in this thesis and would like a decentralized structure for anonymity, the incorporation of MACs might be applicable.

Fabian *et al.* [10] introduce a privacy-preserving P2P data infrastructure for IoT devices based on the Octopus distributed hash table [29] and measured the efficiency of their infrastructure (latency) using simulated networks. While our design does not cover storage issues (the data are only stored locally and discarded immediately after served to the user), it is possible to extend our design by adapting their solution.

Apthorpe *et al.* [4] review privacy vulnerabilities in encrypted IoT traffic of four commercial IoT services. In addition, they develop a strategy to infer consumer behavior from rates of IoT traffic, which enables a passive network observer to retrieve information even when the traffic is encrypted. In this thesis, we use two strategies to avoid such traffic analysis. Firstly, when possible, we use Tor to obfuscate the network location and traffic of IoT devices. Secondly, we cover the traffic to make traffic analysis more difficult.

There are also work focusing on the security of pairing devices. Au [5] perform an analysis of the permission of Android, and propose a tool to analyze the security issues of permission specification. Egele [9] develop program analysis techniques to automatically check programs on cryptographic misuse (whether the cryptographic APIs provides typical cryptographic notions of security, e.g. IND-CPA). In Section 4 of this thesis, we implement an example Android app, and strictly follow the correct usage of permissions and cryptographic functions to prevent security issues.

2.2 Anonymity and IoT

Moving towards anonymous communication for IoT systems, there have been approaches utilizing onion networks. Hiller *et al.* [11] propose a mechanism to tailor onion routing to IoT by bridging the protocol incompatibilities and securely offloading expensive computations to an external server owned by the IoT device owner. Their work focuses on solving problems of incompatible protocols and constrained resources. While our approach proposes a framework for privacy-preserving IoT functionality, Hiller *et al.*'s optimization is also applicable to our approach to provide a more generalized design for privacy-preserving IoT systems.

Hoang *et al.* [13] discuss the challenges and benefits of using the Tor network to secure smart home appliances. They list several vulnerabilities that IoT users face and show how Tor-based communication can help users protect their privacy. This thesis presents a concrete privacy-preserving design for generalized IoT devices and implementation for a specific type of IoT device (an Internet-enabled video doorbell).

Focusing on enhanced security communication for IoT addressing and connectivity, Baumann *et al.* [6] discuss how utilizing Tor benefits in environments behind firewalls, proxies and NAT. They propose a prototype implementation of an Internet-enabled 3D printer using a RaspberryPi as the hardware. Our approach provides a more general framework for IoT security and privacy, and our use case covers a different category of IoT device.

2.3 Background on Tor

The Tor network, which was developed in the 1990s and deployed in 2002, is an overlay protocol to route traffic through multiple servers and encrypt it each step of the way ([14], [7]). By directing internet traffic through the onion network, which consists

of several thousand volunteer overlays, Tor conceals users' locations and usage from adversaries.

Tor allows the creation of *Hidden Services*, which conceal the network locations of Internet services. Such services are only reachable within the onion network by users running the Tor client. Hidden services are identified by *.onion* addresses.

Several features of Tor make it attractive for privacy-preserving IoT communications. Because the nature of Tor requires all peers to be connected, Tor maintains the connections after they are initiated to build the circuits. As a side-effect, all peers can receive data regardless of whether they are behind a firewall or NAT. *Fig. 2.1* shows the network structure of a IoT system adopting the onion network. In such a system, both the IoT device and the end-user device keeps an outgoing TCP connection with onion relays, and thus does not require a static IP address.

These features, including providing NAT piercing and prevention of DoS attacks, are essential for home IoT devices (as home IoT devices are usually located in typical home networks, which tend to be behind NAT). Furthermore, the system of Tor is actively maintained and improved.

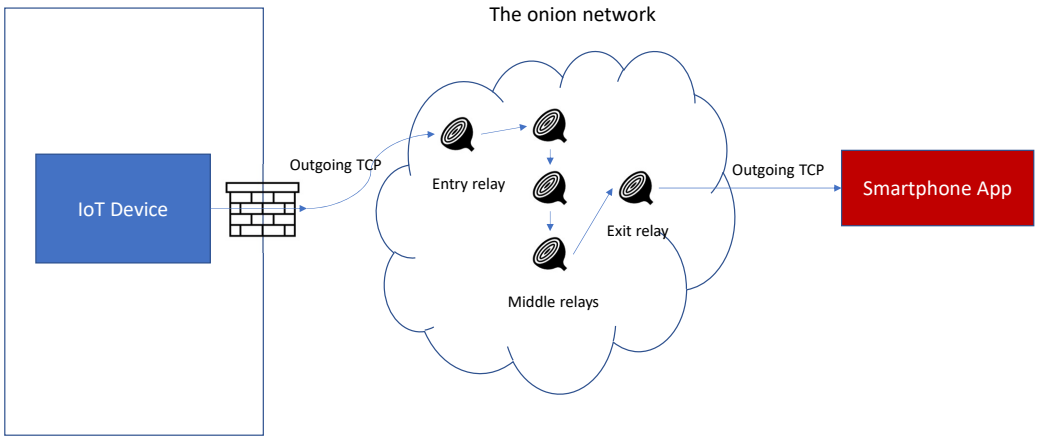

Figure 2.1: Network Structure of onion-IoT

Chapter 3

A privacy-preserving framework for IoT devices

In this chapter, we present our primary contribution, a privacy-preserving framework for generalized IoT devices. We propose communication and attacker models and state security and usability features of the framework.

3.1 Communication models

In this thesis, we consider only one communication model. In our model, the IoT devices interact with end-user devices (e.g., smartphones). The end-user devices have direct access and can remotely control the IoT device.

It is worth mentioning that there exists another communication model for IoT systems. IoT devices sometimes need to communicate with each other to enable automatic workflows. However, we consider the first model to be primary in IoT systems and focus on it. We briefly discuss the IoT-to-IoT scenario in Chapter 5.

Fig. 3.1 and *Fig. 3.2* shows the idea of the two communication models we discussed above.

3.2 Threat model

Our analysis assumes an active network threat model where attackers are located both inside and outside the home network. The attackers may have capabilities similar or the same to an ISP.

Figure 3.1: Communication model: IoT-user

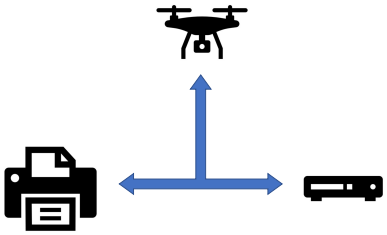

Figure 3.2: Communication model: IoT-IoT

Traditionally, IoT devices run globally searchable services [3]. Consequently, IoT devices are exposed to potential vulnerabilities and attacks. Attackers can discover potentially vulnerable devices by discovering them through IoT search engine (e.g. Shodan) or using whole-Internet scanning tools (e.g. ZMAP [30]).

Furthermore, mobile devices running paired service applications with home IoT devices usually access the Internet via untrusted networks, cellular networks, or free Wi-Fi hotspots. In such cases, on-path attackers can capture and inspect data packets, retrieve sensitive data or even locate and attack the communication's endpoint.

In the following, we discuss potential attackers, their capabilities, and our system's security guarantees.

3.2.1 Actors and capabilities

We consider the following two types of adversaries:

In-network adversaries are adversaries who have access to the users' home network. Such adversaries may eavesdrop and identify traffic from particular devices in the user's home, including the doorbell device and router.

Out of network but on-path adversaries are adversaries located outside the user's home network (and have no access to it). Such adversaries cannot differentiate between devices in the network but can eavesdrop, analyze, and modify the user's traffic in aggregate.

Furthermore, the adversary can obtain and analyze IoT devices and client devices. They may inspect programs and source code and use their copy of client apps to attempt to access the user's device. Also, they may have access to the system's adapted push notification service, which means they can eavesdrop and/or modify the packets or send packets to the user's phone at will.

Service adversaries are adversaries running third-party services (e.g. push notification services) adopted by the IoT system. Such operator should not be able to enumerate all IoT users or know when all events occurred.

3.2.2 Security and Privacy Guarantees

Our framework offers the following security and privacy guarantees:

- **Strong authentication**: The device should only be accessible to authorized clients. *Out-of-network adversaries* should not have access to the data (including the history of usage and data captures by the sensor) regardless of the measurements they take.

- **Anonymity on both sides**: The communication between the client and home IoT device should be anonymized. *Out-of-network adversaries* should get iden-

tical information (including IP address and physical address) of neither the client nor IoT device.

- **End-to-end security**: The traffic between the client and home IoT devices should be end-to-end encrypted. Neither *In-network adversaries* nor *Out-of-network adversaries* should be able to inspect into and read the context in the packets transmitted.

- **Attack resistance**: The attack surface against the IoT devices should be small (e.g., resists DoS attacks).

- **Resistance to traffic analysis**: All types of adversaries should not be able to obtain the user's sensitive information (including but not limited to use history of the IoT device, time of events) by analyzing the encrypted traffic.

3.3 Features

In addition to the security guarantees, the system should have a few usability features:

- **Access from end-user devices.** The system should be accessible from an end-user device. The user should be able to control and retrieve data from the device remotely.

- **Support for NAT-piercing and dynamic IP.** For compatibility with typical home networks, the system should work on devices located behind firewalls, gateway routers or are otherwise normally inaccessible from the outside Internet. Similarly, our framework do not assume static IPs and is compatible with home networks with dynamic IP addresses.

- **Decentralization.** Critically, the system should try to avoid all points of decentralization. The user should not register to third-party services or external websites to have the service running or retrieving data. This eliminates centralization of data, and thus protects users' sensitive information.

- **Real-time data transmission.** The system should let its user retrieve data in real-time and send notifications to the end-user device (e.g. a smartphone) whenever the IOT devices wishes to alert the users.

3.4 Constructing a Privacy-Preserving Framework for IoT Devices.

To achieve the security guarantees described above, we make the approach to introducing the Tor network in the communication between home IoT devices and end-user devices. The Tor network, a common approach for achieving anonymity[8], has the following features, which makes it a good choice for privacy-preserving systems:

Achieving strong authentication and end-to-end confidentiality. This task is non-trivial since IoT devices have limited interfaces. The framework tackles this by enabling a pairing phase in which smartphone devices can pair with the IoT device and exchange key material. The pairing phase must be manually initiated on the IoT device, and should be resistant to man-in-the-middle attacks. One instantiation of this (see next Chapter) is to have the IoT device set up an mDNS service, which makes it possible for a smartphone in the same network to connect to share key material. Once key material is shared, all data sent and received by both the IoT device and the smartphone application should be end-to-end encrypted using standard protocols (e.g. TLS). Additional methods for enabling strong authentication are described below (see "Attack resistance").

Resistance to traffic analysis. A key feature of the framework is that it (1) obscures the use of the IoT device and (2) provides resistance to traffic analysis from potential on-path adversaries. Both are achieved by tunneling all traffic from the IoT through an anonymity network, such as Tor. Additionally, covertness can be achieved by using Tor pluggable transports [23], which further conceal the anonymity network's use. This makes it difficult for an adversary to even identify that the monitored user's home network contains the IoT device, given that Tor has a wide variety of uses well beyond obscuring IoT traffic.

Push notification services present another opportunity for traffic analysis. In traditional IoT systems, the IoT device sends a push notification through an operator (e.g. Google on Android and Apple on iOS) in order to alert the owner of the device. This allows the push notification operator to learn (1) the users who have the IoT device (by virtue of forwarding the alerts to their smartphones), (2) when such alerts occurred, and (3) if not encrypted, the contents of the notification. We protect against all three forms of information leakage by avoiding centralized push notification services, and instead, sending push notifications directly (or more precisely, through Tor) from the IoT device to the smartphone. In the next Chapter, we show that the power and data cost of such an operation are surprisingly small.

Attack resistance. Our framework requires the IoT device to operate as a Tor onion service (previously called *hidden services*). Onion services are only accessible via the Tor network. This provides strong resistance to denial-of-service attacks, since Tor does not support voluminous amounts of traffic (since it is slow) as all traffic must traverse a series of (unfortunately overloaded) relays. Additionally, the use of hidden services also further supports strong authentication, since onion services optionally

support authentication – meaning that the service is only accessible to clients who possess the correct keys.

Resistance to enumeration. The use of onion services also provides resistance to enumeration. Traditional IoT devices may be (and often are) discoverable on the Internet, and indeed, search engines such as Shodan [20] provide queryable interfaces for identifying particular IoT devices. This poses a significant security threat, since a single vulnerability in an IoT device could easily be used as a mechanism to construct a (potentially large) botnet. Our framework resists such enumeration through the use of onion services. Onion services cannot be easily enumerated [31] [7] since connecting to them requires knowledge of both the .onion URL and, in our case, the credentials for accessing the onion service.

Chapter 4

Case-study: A privacy-Preserving Video Doorbell

This chapter presents a case study that focuses on the design and implementation of a specific type of IoT device - an Internet-enabled video doorbell using our privacy-preserving framework. We assume the communication model to be the user-to-IoT model, that is, the IoT device (video doorbell) is located in the user's home network and communicates (potentially indirectly) with the end-user device (smartphone). The goal of this case-study is to support a feature-rich IoT offering while providing strong security and privacy properties.

4.1 Features and system design

In addition to the security guarantees described in the last chapter, video doorbell systems have unique features. Here we present the design of the specific system.

4.1.1 Features

We consider there should be five prominent use cases in the system:

- **Registration.** The user should be able to register a mobile device to the doorbell. Only trusted devices can access the data and manage settings.
- **Video streaming.** The user with a registered device should be able to access video captured by the camera at any time they wish.

- **Face detection and push notification.** Whenever a visitor appears in front of the camera and/or presses the physical doorbell button, all users registered to the doorbell should receive a push notification message on their smartphones.

- **Setting management.** The user should be able to manage settings, including changing face detection threshold, recording voice to be played, removing trusted devices, etc.

- **Two-way conversations.** The system should enable two-way conversations, in which the visitor can directly speak to the doorbell user and the user can play pre-recorded voice in response. The pre-recorded voice acts as an alternative way of answering to decrease the high latency introduced by Tor.

Fig. 4.1 shows the use case diagram of the system.

We describe the specifics of our implementation in more detail below. At a high-level, our video doorbell consists of a computing platform (Raspberry Pi), a video camera and an external sound card.

4.1.2 Network architecture

Fig. 4.2 shows the network architecture of the system. Both the client and server communicate using Tor to prevent an adversary from trivially identifying the devices' traffic. Additionally, we configure Tor on the doorbell side to use Snowflake pluggable transport. Snowflake is an add-on for Tor that tunnels Tor traffic through WebRTC [19], a protocol that is popular for videoconferencing (e.g. Google's Hangouts or Jitsi). By using Snowflake, an adversary that monitors the communication from the user's home cannot easily identify the use or presence of the IoT doorbell, since its traffic is disguised as WebRTC flows.

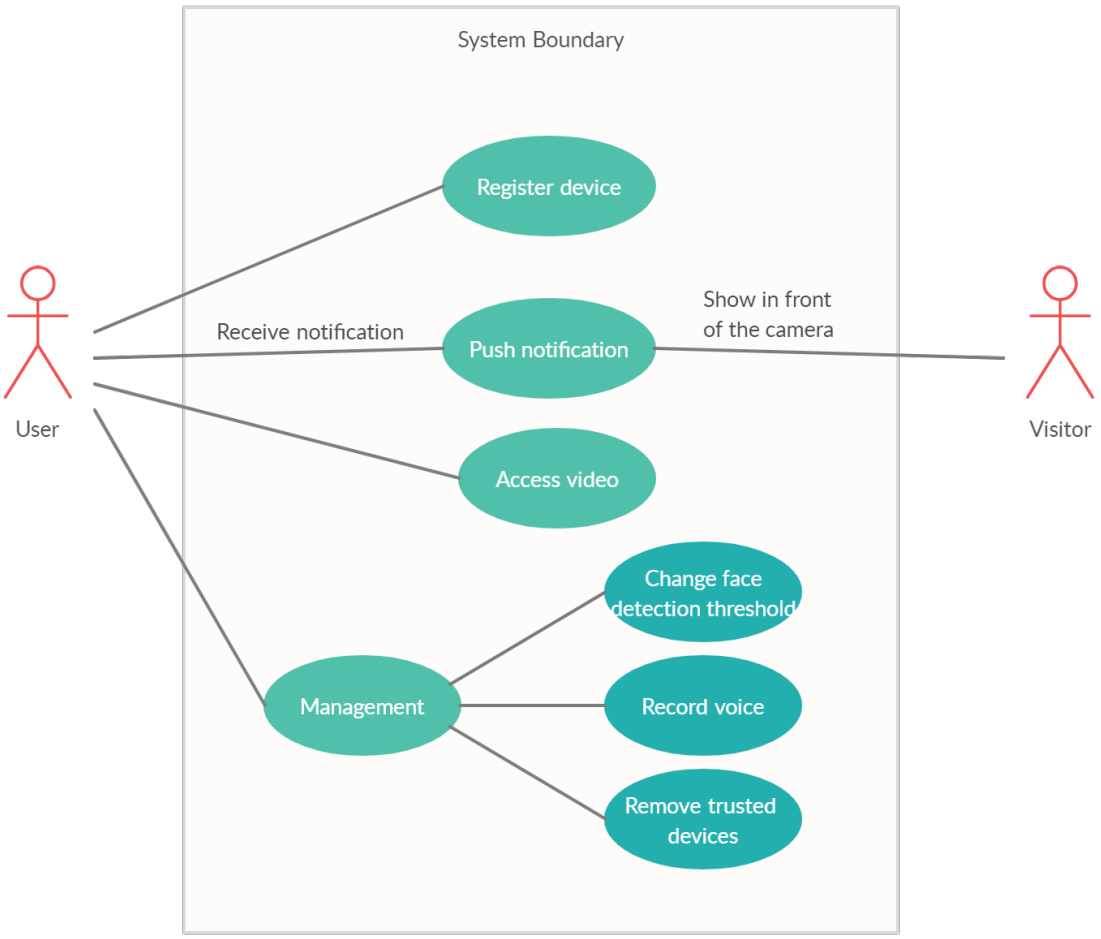

Figure 4.1: Use case diagram

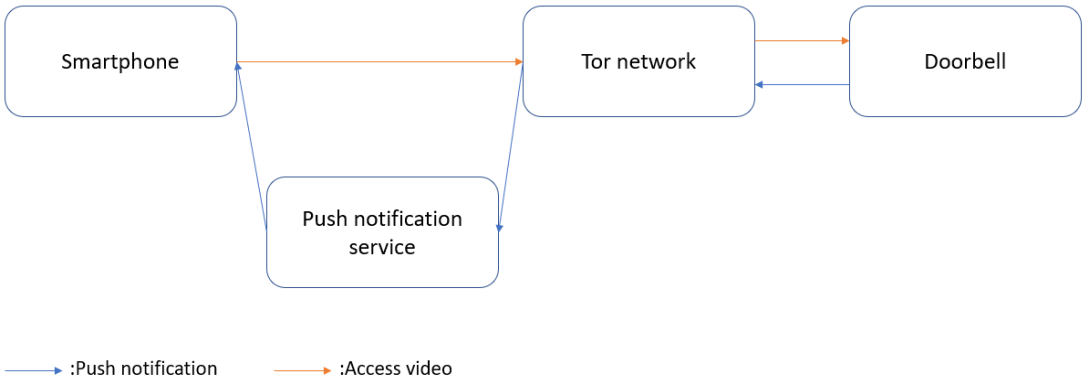

Figure 4.2: Network architecture diagram

4.1.3 Operation

Registration. The very first stage of using the system is the registration process. To have their devices registered on the doorbell, the users should first connect to the same wireless network with the doorbell. Then the app should allow the user to register easily by searching the doorbell using mDNS (Multicast DNS, also known as mDNS, is a protocol resolves hostnames to IP addresses in small networks, like local home networks, which do not include a local name server.). During the registration process, the smartphone app performs a key exchange with the doorbell and receives randomly generated credentials for the user to set up in Tor. We use Orbot, an open-source implementation of Tor, on the smartphone. Currently, our implementation supports Android, although providing support for iOS should be straightforward.

Playing video and authentication. Our IoT doorbell allows the user to observe activities that occur at their doorstep by activating the doorbell's video camera and streaming the video to the smartphone app. As soon as the user presses the PLAY button in the app, it will work with Orbot and send an RTMP PLAY request to the video server running on the doorbell. The video server will then forward an authentication request (in the form of an HTTP GET request) to the authentication server (running on another port on the doorbell). The server should serve video to the user through Tor if the authentication succeeds (i.e., the app is registered and has not been revoked) and shut down the connection otherwise. *Fig. 4.3* shows the flow of playing video and authentication. As described above, there are five components in the flow: the user (*user*), the smartphone app (*app*), the Orbot app (*Orbot*), the video server (*videoserver*) and the authentication server (*authserver*).

Face detection, push notification, and answering the door. The doorbell constantly detects if there is a human face appearing in front of the camera. Whenever

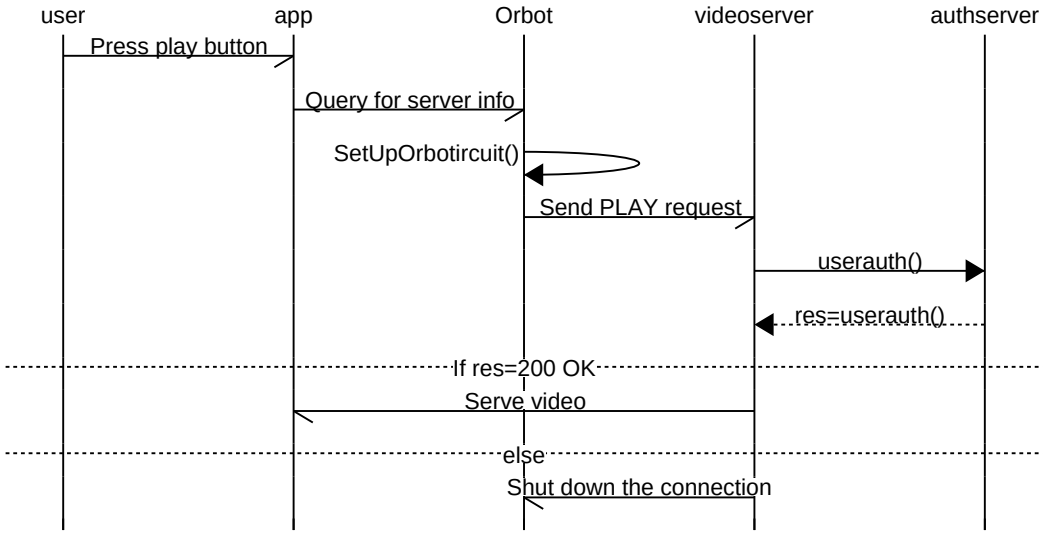

Figure 4.3: Sequence diagram: Video streaming

a visitor appears, the system should send a notification to the user's devices through Google's push notification service, Firebase Cloud Messaging (FCM).

By clicking on the notification message on her phone, the user should be able to access the video immediately and choose to play pre-recorded audio on the doorbell device to the visitor. *Fig. 4.4* shows the flow of face detection, push notification, and answering the door.

4.2 Face detection

A core feature of our privacy-preserving IoT framework is the avoidance of points of centralization. Although many cloud-based face detection services exist (e.g. Google's Firebase MLKit), we restrict our IoT doorbell to use on-device face detection. (We also remark that our devices does not perform face textitrecognition, which is a seperate task that attempts to identify the individual who the face belongs to.)

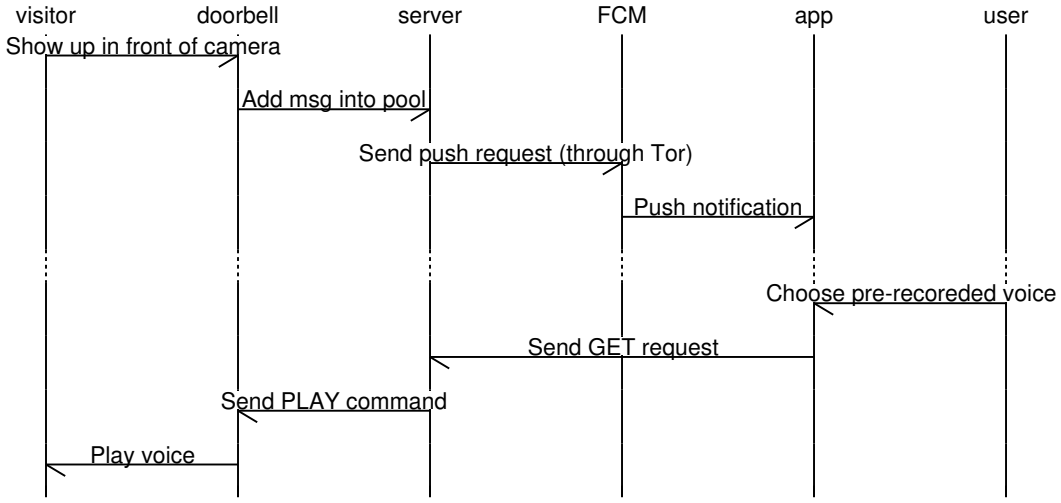

Figure 4.4: Sequence diagram: Push notification

For the face detection task, we use the pre-trained CascadeClassifier [28]. Cascade Classifiers based on Haar-like features were introduced in 2001 and are still widely used in face detection [24]. Pre-trained Cascade Classifiers have a short execution time and small calculation load, making them a good choice for on-device calculations on IoT devices with low computational power.

We sample 6 frames per second in the actual implementation, transform them into gray-scale images, and feed them to the pre-trained classifiers. The detector is disabled for 5 seconds (which also matches the push notification interval) after a successful detection to prevent abusing resources.

4.3 Device specifications

We configured a RaspberryPi 4 model B as the doorbell device. The RaspberryPi 4 has a built-in Wi-Fi antenna, and we combined the device with a camera module and

external sound card. The RaspberryPi ran the Raspbian Jessie OS, which is a version of Debian Linux optimized for the RaspberryPi platform.

The RaspberryPi 4 model is equipped with 2GB of RAM and a 4-core CPU of 1.5 GHz, which provides sufficient computation power for our requirements.

4.4 Push notification

As with other IoT doorbell systems, our implementation alerts the user whenever an individual approaches their house. To notify the user whenever a visitor is present in front of the camera, the system adopts Google's Firebase Messaging Service.

We designed two types of messages: **BELL** (sent when someone pushes the doorbell device's physical button) and **DOOR** (sent when someone comes in front of the camera). The packet includes a message (including the type mentioned above), a timestamp, and the user's instance token. The message and timestamp are encrypted using AES-256-GCM with pre-shared keys (which are shared in the registration process described later).

In order to prevent the third-party service provider (i.e., Google) from obtaining sensitive information (e.g., the comings and goings of visitors) by analyzing timing information, we further cover the traffic using another message type **DUMMY**, and send messages in a fixed interval of 5 seconds. The dummy packets have very similar structures to the normal ones, but have different types to be recognized by the client app. We note that **BELL**, **DOOR** and **DUMMY** packets are all encrypted and appear indistinguishable to the push service provider. We ensure that the system sends a packet every 5 seconds. However, it is worth mentioning that there may be potential optimizations on both the traffic cover mechanism and the push notification system.

We will discuss them in Section 5.1 and 5.2, and show a simple proof-of-concept of a fully-decentralized push notification system.

4.5 Video streaming

The doorbell device captures video using the RaspberryPi's camera module and audio through an external sound card. The video captured is encoded and served in flv format through HTTP. We chose to encode the videos in flv format because flv support is common in many operating systems and thus support future portability to other platforms.

Our system adopts two layers of authentication on top of video streaming. The first layer of authentication is *onion authentication*. Provided along with Tor, it requires the user to have specific credentials set up in their Tor (Orbot) client. The onion host refuses to connect if the client does not have such credentials. The second layer of authentication is the *RTMP authentication*. In order to access the video, the user will have to add credential arguments in addition to their RTMP PLAY request. *Fig. 4.5* shows the components of the video serving URL. In the URL, *appname* and *streamname* are Nginx settings and of the user's choice. *usertoken* is a random number securely generated from the client app (upon the first launch), and the following equation calculates userpassword:

$$userpassword = HMAC(seed, usertoken)$$

where the HMAC is based on SHA256.

The server checks if *all* credentials match and shuts down the connection if any are incorrect. The *seed* is a per-device secret, and its generation and use is explained in more detail in Section 4.6.1.

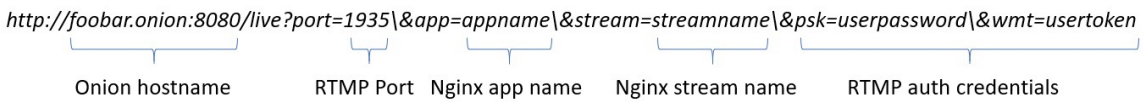

Figure 4.5: An example of video serving URL

4.6 Android App

We implemented an Android App to pair with the video doorbell service. The app runs on the end-user smartphone device and grants users access to the doorbell device and data. The app has the following primary functions:

4.6.1 Registration

For registration, the client sends JSON-formatted data in an HTTP POST request to port 8080 of the server (running on the doorbell device). This occurs over WiFi in the user's home. The server will then respond with another JSON-formatted data, including the seed (randomly generated 16-bit integer) for calculating the secret keys, the onion hostname for accessing the video service, and the onion authentication cookie. The app will then automatically send a configuration string (including the authentication cookie) required by Orbot to the user's clipboard for her to set up the service quickly.

Fig. **??** shows the main (registration) GUI of the app.

4.6.2 Media player

The app has the VLC player integrated for video streaming. This allows users to see what is occurring at their doorstep. Working with Orbot, the media player streams the video through the onion network, which prevents adversaries from eavesdropping

on the transmitted video. The use of Snowflake (as described in Section 4.1.2 also disguises the video stream.

4.6.3 Two-way communication

The media player streams both video and audio and thus enables the user to see and hear visitors. Meanwhile, the user has the option to play pre-recorded voice samples (which can be managed through the app as well, as described in Section 4.6.4) to communicate with visitors. We envision supporting real-time two-way communication as a future feature.

4.6.4 Device management

The doorbell serves a webpage for the user to manipulate device settings, including changing the face detection threshold, revoking authentications, recording voice, etc. The management page is only accessible in the local network (that is, the user's smartphone device must be in the same wireless network as the doorbell device to manipulate settings) and requires a password for access.

Fig. **??** shows the preference page, and *Fig.* **??** shows the token management (where user can revoke trusted devices) of the app.

4.7 Performance Evaluation

In this section, we would like to evaluate the performance of our system by analyzing the following factors:

- Streaming latency;
- Energy consumption;

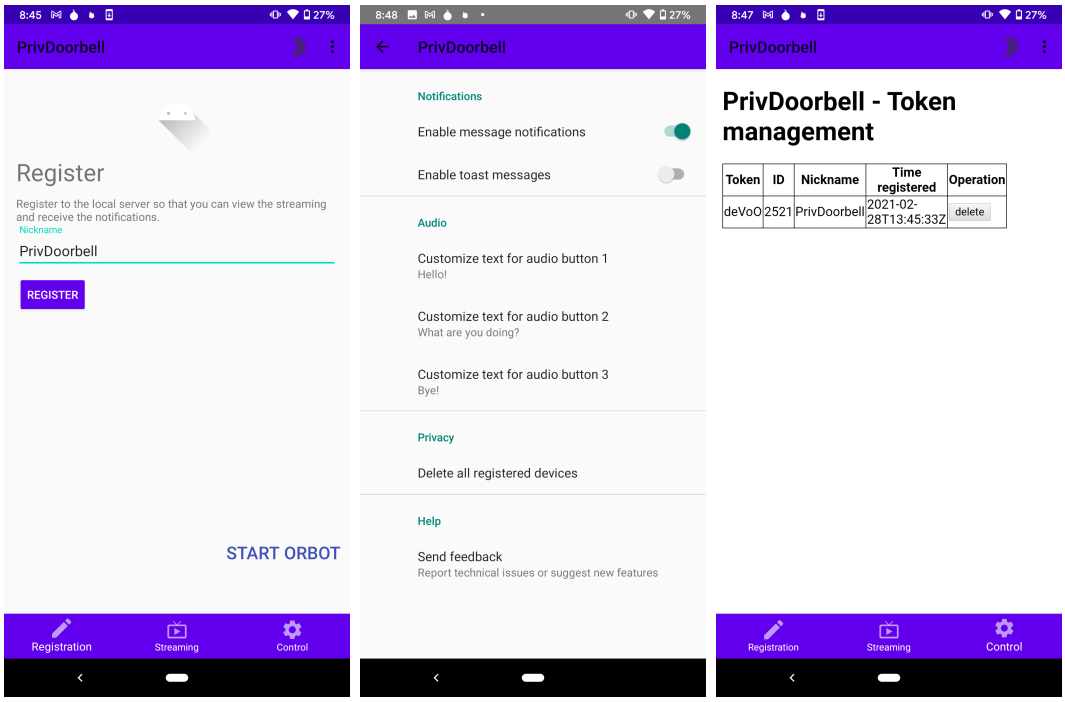

Figure 4.6: Android App GUI Examples

- Bandwidth;
- Notification latency;

A Google Pixel 3a XL is used as the client device.

4.7.1 Streaming latency

We measure our system's latency as the time difference between the user's pressing on the Play button and the first frame's appearing. *Fig. 4.7* shows the time taken for 100 samples. We divide the time into four phases.

- Circuit: The *Circuit* phase is from when the app starts processing the user's request to when it receives the server's information. In this phase, the app

queries the proxy (Orbot) about the onion URL. Orbot will then try to establish a circuit connection between the user device and the server. In practice, the time taken in this phase varies, as sometimes there is an existing circuit. If Tor has to create a new circuit, this phase typically takes a few more seconds. On average, this phase takes 2.355 seconds with a standard derivation of 3.07 seconds.

- Transmission: The *Transmission* phase is from when the app sends request to the server to when the app hears back from the server. As we are using RTMP authentication, the server should return an HTTP response code 200 (indicating success) if the client provides correct credentials. The video transmission will begin immediately after the HTTP response. In average, this phase takes 0.599 seconds with a standard derivation of 0.089 seconds.

- Preparation: The *Preparation* phase is from when the app receives the HTTP response from the server to when it starts filling the buffer. In this phase, the media player initializes its components and gets ready for playing the video. In average, this phase takes 3.878 seconds with a standard derivation of 0.715 seconds.

- Buffering: The *Buffering* phase is when the app fills in its buffer. In this phase, the media player fill a few frames into the buffer (of a fixed size) for the decoder to decode. The decoder will decode the frames in milliseconds, and thus we can consider the video being played immediately after this phase. In average, this phase takes 0.046 seconds with a standard derivation of 0.006 seconds.

On average, the whole process takes 6.867 seconds with a standard derivation of 3.141 seconds. *Fig. 4.8* shows how much time each phase takes. While we believe a 7 second response time is certainly noticeable, we also posit that such a delay is reasonable for our video doorbell.

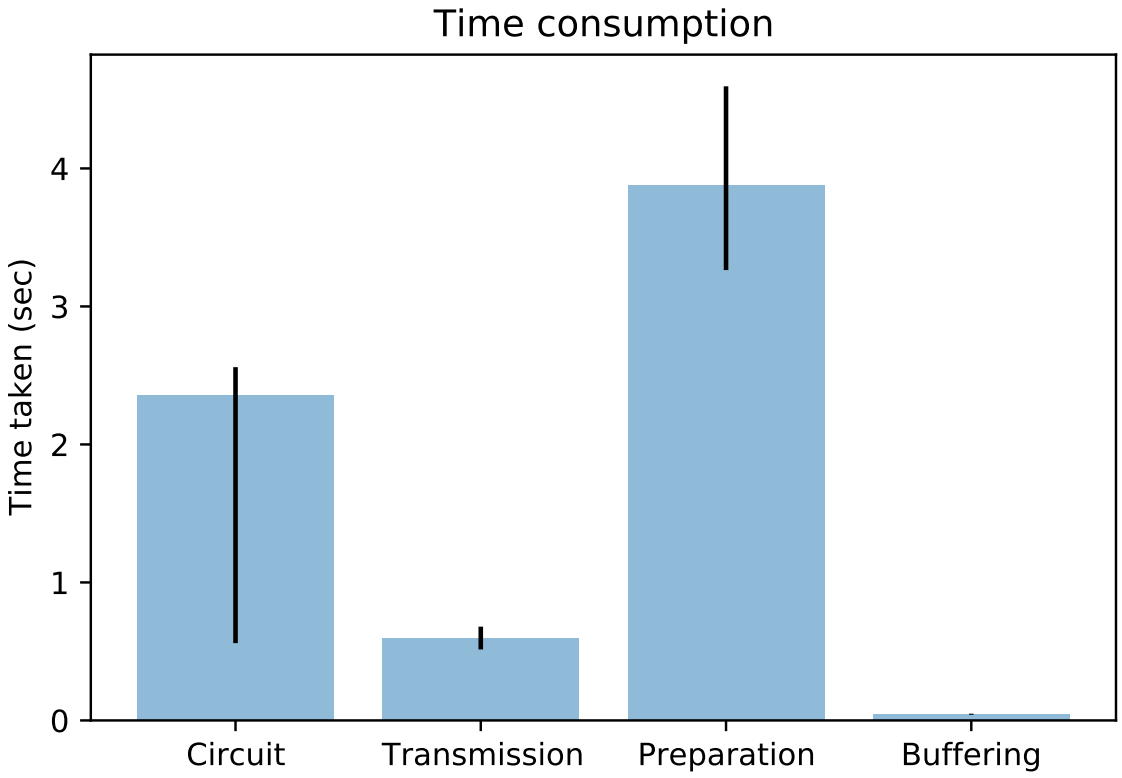

Figure 4.7: Time consumption of the four phases

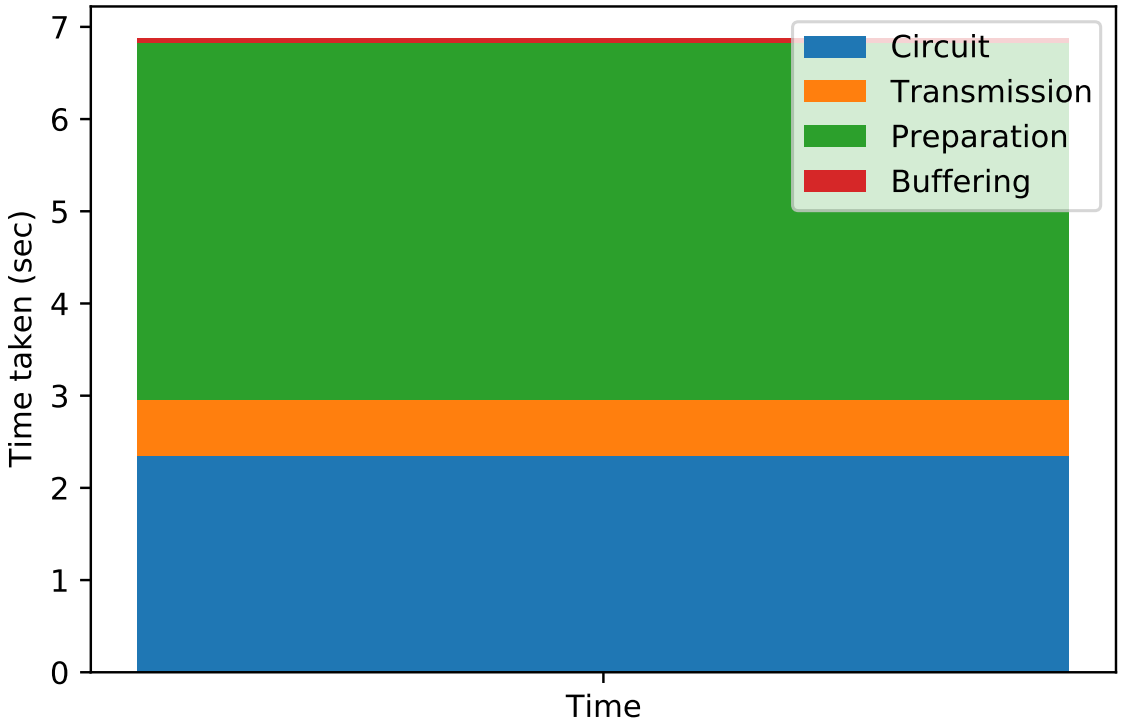

Figure 4.8: Time consumption of the four phases (percentage)

4.7.2 Energy consumption

We evaluate the battery consumption of our app using the system measured data. The app consumes 6% of the device's battery after actively running in background for 6 days, 11 hours and 50 minutes (83.83 hours). The device has a battery size of 3700 mAh, and therefore the app roughly consumes 2.65 mAh for every hour running. We consider it reasonable for an app which receives and pushes real-time notifications.

4.7.3 Bandwidth

The outgoing traffic from the doorbell devices is 71 kb/s by average, after the video is compressed and encoded. For comparison, the raw video has the a of 410 kb/s. When not streaming the video, the doorbell's bandwidth cost is negligible.

4.7.4 Video and audio quality

The system supports videos up to 1024x768 at 6 fps and audio with a sample rate of 44.1kHz.

4.7.5 Notification latency

We measure the notification latency by measuring the time difference from when the message is sent to Firebase Messaging server from the doorbell to when the message is received on the client app. The average latency of 250 consecutive samples is 0.478 seconds. Fig. 4.9 shows the CDF (cumulative distribution function) vs. latency.

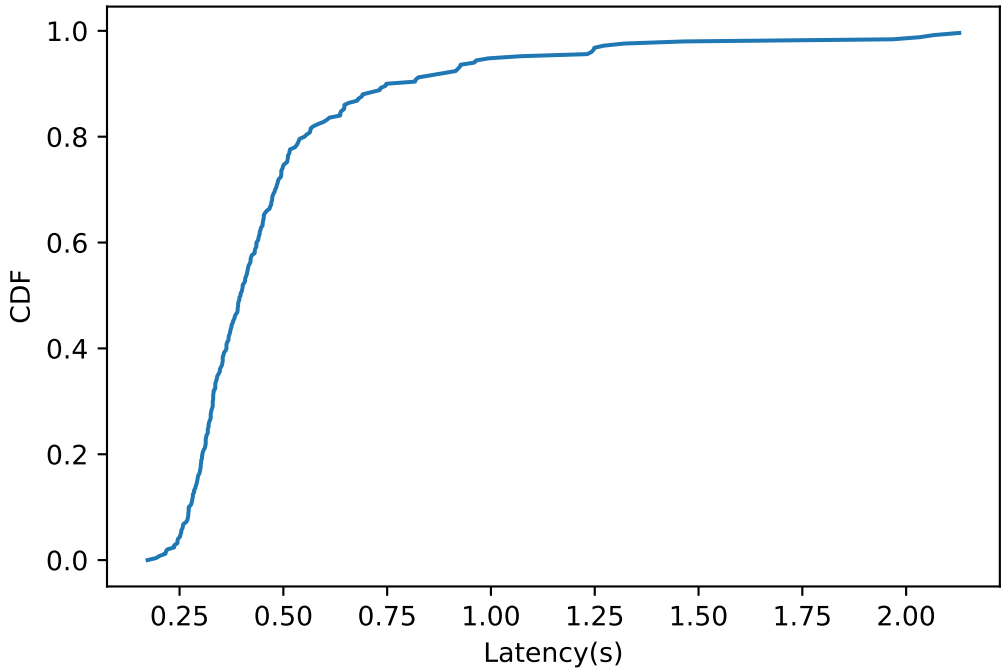

Figure 4.9: Latency of notification - with Tor

4.7.6 The cost of using Tor

Finally, we measure the cost of using Tor. By its nature, Tor adds latency to the system and we would like to know the exact impact. We measure the following time and compare the difference between:

- Streaming latency
- Notification latency

Streaming latency We described the streaming latency of our system in Section 4.7.1. For a simple comparison, launching the streaming takes 2.240 seconds on average.

Notification latency As we described in Section 4.7.5, the latency of push notification with Tor is 0.478 seconds on average. We further conduct experiment where Tor is not in the middle. *Fig. 4.10* shows the difference of the service with Tor ("with-Tor") between that without Tor ("vanilla"). The dotted green line denotes for the latency of vanilla service, and the blue full line denotes for the latency of with-Tor service. The median of with-Tor service is roughly twice as the median of vanilla service.

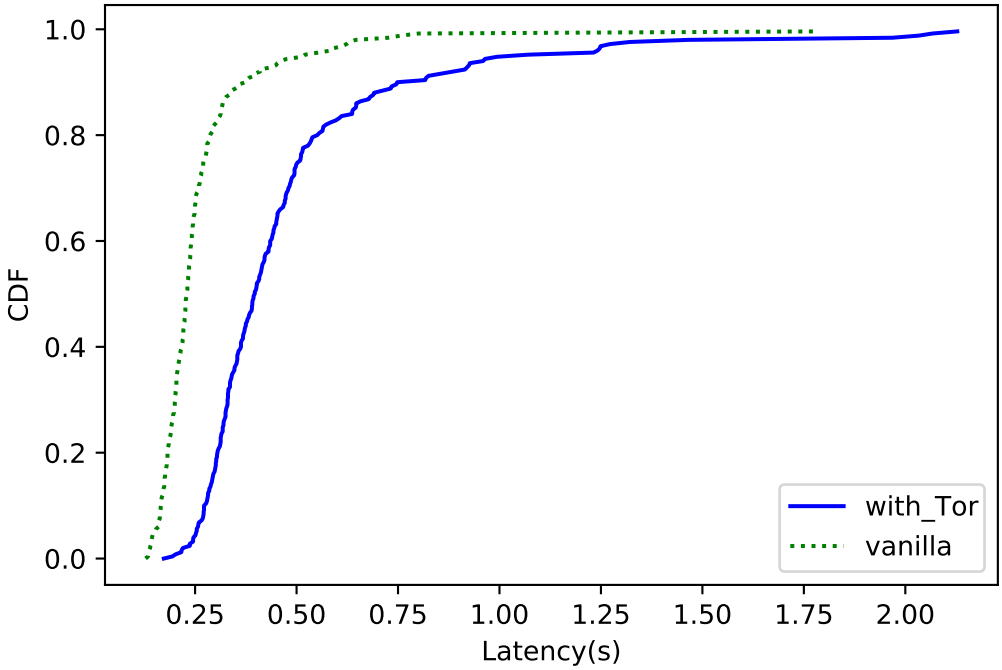

Figure 4.10: Latency of notification - with-Tor vs. vanilla

Chapter 5

Discussion and Open Questions

5.1 Traffic cover

In the case study, we covered the traffic of push notifications by padding dummy packets and sending packets constantly (every 5 second). While constant padding is a safe countermeasure against traffic analysis, the scheme may consume much more power than necessary since the interval of people visiting and thus non-cover message is usually far larger than 5 seconds.

Prior research has proposed efficient ways of padding traffic. Perhaps most notably, Juarez *et al.* [15] proposed an adaptive padding algorithm and proved its effectiveness against traffic analysis. We consider it possible to adapt a similar algorithm in our system as a possible future enhancement.

5.2 Decentralized push notification

Ideally, our system would like to avoid all points of centralization and push notification is especially challenging since such alerting service are highly centralized. It is nearly impossible to push a notification with out using *APNS (Apple Push Notification Service)* on an iOS device. It is somewhat easier on Android devices as Google permits the use of third-party push notification services. Unfortunately, most Android push notification services are designed to be 1-to-many systems (that is, one app developer

pushes to many user devices), which makes it a bit difficult for private IoT devices to utilize such service.

In our case study, we used Firebase Cloud Messaging (formerly known as Google Cloud Messaging) as the push notification service provider. FCM is also designed as 1-to-many, and requires credentials for message senders. Therefore, in order to get a private channel for their own message, the users will have to register their own API key, include the key and compile the APK themselves. We consider the inconvenience a shortcoming of our system.

It is worth mentioning that implementing a private channel for push notifications is possible. A naive solution would be keeping a TCP socket open over Tor to receive notifications. Kollmann *et al.* [16] discussed about the possibility and cost of push notifications using Tor, and pointed out that such solution is feasible but would consume more resource than most commercial push notification services.

For a proof-of-concept, we also implemented an experimental option for users to adapt such way of push notifications. In particular, we constructed our own push notification service for Android in which all communication occurs over the Tor network. Although we still reply on centralization (i.e., the push notification server), the use of Tor obfuscates clients' (the doorbells' and Android devices') locations, and prevents the centralized service from associating particular smartphones with doorbells. Further, it prevents the centralized server from easily enumerating the number of devices. We believe that such a privacy-preserving push notification is valuable for a number of uses outside of our IoT doorbell proof-of-concept. Experiments show that keeping the connection active for 4 hours consumes about 2% of the device's battery. While the option is not ready for practical use (as there have been some unimplemented system-level issues, e.g. handling messages when the device is unavailable or the app has exited), it shows the possibility to deploy a fully decentralized push notification

service with our system. We consider the full implementation to be one of the future enhancements.

5.3 Storage

In our proof-of-concept implementation, we did not add in the capability to store user videos. The videos are immediately discarded after being served. We consider it unsafe to store all the videos locally, as an adversary could physically steal or destroy the device. However, video doorbell users usually want to see what happened a few minutes or even a few days ago.

Most commercial video doorbells have applied the solution that they stored the video in their cloud server (and charge the users service fees). In this setting, the user's privacy completely depends on how the provider handles their information.

Another solution would be taking advantage of distributed storage. It is possible for users to form a distributed storage network, taking advantage of Distributed Hash Table [27]. Thanks to DHT's property of being fault tolerant, it makes saving user data among all system users possible. There have been several approaches to apply DHT in IoT devices [10] [21].

A practical problem with our system adapting distributed storage would be the size of data. Unlike most other IoT devices, video doorbells need to store and transmit large chunks of data (i.e. videos), which makes it difficult to implement in terms of fault tolerance.

5.4 IoT devices in untrusted network

In this thesis, we consider the home IoT device to be located in a trusted network (i.e. the home network). However, there are scenarios in which such devices will also

be located in untrusted networks, for example the user is carrying the device while traveling. In such cases, additional attack surfaces and vulnerabilities are introduced as attackers can easily identify the IoT devices, analyze and manipulate the traffic. While such scenarios are not studied in this thesis, we consider it possible to adapt our solution with some additional security measurements.

5.5 IoT-to-IoT communication

There is another communication model where IoT devices communicate with each other. This usually happens when a user has multiple home IoT devices and enables automatic workflows. While such a communication model is not covered in this thesis, it is possible to extend our solution so that each IoT device hosts its own onion service.

Chapter 6

Conclusion

This thesis explores the challenge of designing privacy-preserving systems for home IoT devices. It reviews currently available home IoT devices, their privacy threats and the need for a privacy-preserving framework.

This thesis next proposes a privacy-preserving framework for generalized home IoT devices by combining Tor with IoT systems. We present different attacker models, security guarantees and usability features of such framework.

Then, it presents a case study which focuses on the detailed design and implementation of a specific type of home IoT systems - an Internet-enabled doorbell. We proposed the system design and implemented the system on a RaspberryPi computer. We also did experiments and analysis to argue the implemented system's usability.

Finally, we discuss open questions that are not covered in this thesis, including better traffic cover mechanism, completely decentralized push notification service, distributed storage and different communication models.

Bibliography

[1] Yasemin Acar, Michael Backes, Sven Bugiel, Sascha Fahl, Patrick McDaniel, and Matthew Smith. 2016. Sok: Lessons learned from android security research for appified software platforms. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 433–451.

[2] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. Sok: Security evaluation of home-based iot deployments. In *2019 IEEE symposium on security and privacy (sp)*. IEEE, 1362–1380.

[3] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the mirai botnet. In *26th {USENIX} security symposium ({USENIX} Security 17)*. 1093–1110.

[4] Noah Apthorpe, Dillon Reisman, and Nick Feamster. 2017. A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic. *arXiv preprint arXiv:1705.06805* (2017).

[5] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. 2012. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 217–228.

[6] Felix W Baumann, Ulrich Odefey, Sebastian Hudert, Michael Falkenthal, and Uwe Breitenbücher. 2018. Utilising the Tor Network for IoT Addressing and Connectivity.. In *CLOSER*. 27–34.

[7] Abdelberi Chaabane, Pere Manils, and Mohamed Ali Kaafar. 2010. Digging into anonymous traffic: A deep analysis of the tor anonymizing network. In *2010 fourth international conference on network and system security*. IEEE, 167–174.

[8] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The second-generation onion router*. Technical Report. Naval Research Lab Washington DC.

[9] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. 2013. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 73–84.

[10] Benjamin Fabian and Tobias Feldhaus. 2014. Privacy-preserving data infrastructure for smart home appliances based on the Octopus DHT. *Computers in Industry* 65, 8 (2014), 1147–1160.

[11] Jens Hiller, Jan Pennekamp, Markus Dahlmanns, Martin Henze, Andriy Panchenko, and Klaus Wehrle. 2019. Tailoring onion routing to the Internet of Things: Security and privacy in untrusted environments. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, 1–12.

[12] Stephen Hilt, Vladimir Kropotov, Fernando Mercês, Mayra Rosario, and David Sancho. 2019. The internet of things in the cybercrime underground. *Trend Micro Research* (2019).

[13] Nguyen Phong Hoang and Davar Pishva. 2015. A TOR-based anonymous communication approach to secure smart home appliances. In *2015 17th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 517–525.

[14] The Tor Project Inc. 2021. The Tor Project: Privacy Freedom Online. https://www.torproject.org/about/history/

[15] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. 2016. Toward an efficient website fingerprinting defense. In *European Symposium on Research in Computer Security*. Springer, 27–46.

[16] Stephan A Kollmann and Alastair R Beresford. 2017. The Cost of Push Notifications for Smartphones using Tor Hidden Services. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 76–85.

[17] Selena Larson. 2017. FDA confirms that St. Jude's cardiac devices can be hacked. https://money.cnn.com/2017/01/09/technology/fda-st-jude-cardiac-hack/

[18] Huichen Lin and Neil W Bergmann. 2016. IoT privacy and security challenges for smart home environments. *Information* 7, 3 (2016), 44.

[19] Kyle MacMillan, Jordan Holland, and Prateek Mittal. 2020. Evaluating Snowflake as an Indistinguishable Censorship Circumvention Tool. *arXiv preprint arXiv:2008.03254* (2020).

[20] John Matherly. 2015. Complete guide to Shodan. *Shodan, LLC (2016-02-25)* 1 (2015).

[21] Federica Paganelli and David Parlanti. 2012. A DHT-based discovery service for the Internet of Things. *Journal of Computer Networks and Communications* 2012 (2012).

[22] Kari Paul. 2019. Dozens sue Amazon's Ring after camera hack leads to threats and racial slurs. https://www.theguardian.com/technology/2020/dec/23/amazon-ring-camera-hack-lawsuit-threats

[23] Khalid Shahbar and A Nur Zincir-Heywood. 2015. Traffic flow analysis of tor pluggable transports. In *2015 11th International Conference on Network and Service Management (CNSM)*. IEEE, 178–181.

[24] Ali Sharifara, Mohd Shafry Mohd Rahim, and Yasaman Anisi. 2014. A general review of human face detection including a study of neural networks and Haar feature-based cascade classifier in face detection. In *2014 International Symposium on Biometrics and Security Technologies (ISBAST)*. IEEE, 73–78.

[25] Ilia Shumailov, Laurent Simon, Jeff Yan, and Ross Anderson. 2019. Hearing your touch: A new acoustic side channel on smartphones. *arXiv preprint arXiv:1903.11137* (2019).

[26] Tianyi Song, Ruinian Li, Bo Mei, Jiguo Yu, Xiaoshuang Xing, and Xiuzhen Cheng. 2017. A privacy preserving communication protocol for IoT applications in smart homes. *IEEE Internet of Things Journal* 4, 6 (2017), 1844–1852.

[27] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review* 31, 4 (2001), 149–160.

[28] Paul Viola and Michael Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, Vol. 1. IEEE, I–I.

[29] Qiyan Wang and Nikita Borisov. 2012. Octopus: A secure and anonymous DHT lookup. In *2012 IEEE 32nd International Conference on Distributed Computing Systems*. IEEE, 325–334.

[30] Stefan Wiemer. 2001. A software package to analyze seismicity: ZMAP. *Seismological Research Letters* 72, 3 (2001), 373–382.

[31] Philipp Winter, Anne Edmundson, Laura M Roberts, Agnieszka Dutkowska-Żuk, Marshini Chetty, and Nick Feamster. 2018. How do Tor users interact with onion services?. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 411–428.

[32] Nicky Woolf. 2016. DDoS attack that disrupted internet was largest of its kind in history, experts say. `https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet`