Title

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

Author, B.S.

Washington, DC
May 5, 2005

Title

Author, B.S.

Thesis Advisor: First I. Last

Abstract

This is the abstract, a brief summary of the contents of the entire thesis. It is limited to 350 words.

Index words: word1, word2, word3

Dedication

The Dedication is optional.

Acknowledgments

The author would like to thank...

Preface

A preface is not an introduction, and most theses do not need them.

Table of Contents

Acknowledgments v

Preface vi

Chapter

1 Introduction 1
- 1.1 Background on IoT devices 1
- 1.2 Motivation: need for privacy-preserving framework 1
- 1.3 Overview of approach 1
- 1.4 Scientific questions and contributions 1
- 1.5 Background on Tor 1

2 Related work 3
- 2.1 Papers on IoT security 3
- 2.2 Papers on anonymity 4

3 A privacy-preserving framework for IoT devices 6
- 3.1 Communication models 6
- 3.2 Threat model 6
- 3.3 Feature 8
- 3.4 Methodology 9

4 Case-study: A privacy-Preserving Video Doorbell 11
- 4.1 System design 11
- 4.2 Specs of the devices 16
- 4.3 Face detection 16
- 4.4 Push notification 16
- 4.5 Video streaming 17
- 4.6 Android App 18
- 4.7 Experimental study 20

5 Discussion and Open Questions 27
- 5.1 Traffic cover 27
- 5.2 Decentralized push notification 27
- 5.3 Storage 28
- 5.4 IoT devices in untrusted network 29
- 5.5 IoT-to-IoT communication 30

6 Conclusion 31
Appendix
A First Appendix 32
B Second Appendix 33
Bibliography 34

List of Figures

4.1 12
4.2 13
4.3 14
4.4 15
4.5 15
4.6 18
4.7 19
4.8 19
4.9 19
4.10 22
4.11 23
4.12 25
4.13 26

List of Tables

Chapter 1

Introduction

1.1 Background on IoT devices

1.2 Motivation: need for privacy-preserving framework

1.2.1 Privacy threats of IoT devices

1.2.2 Challenges

1.3 Overview of approach

1.4 Scientific questions and contributions

1.5 Background on Tor

The Tor (The Onion Router) 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 ([2], [7]). By directing internet traffic through the onion network which consists of several thousands volunteer overlays, Tor conceals users' location and usage from adversaries.

Tor allows the consumption of *Hidden Services*. Such services are only reachable within the onion network by users running the Tor client. Hidden services are identified by *.onion* addresses.

The security features of Tor makes it an attractive solution for IoT communications. Besides, Tor has a few great usability features including NAT piercing and DoS

prevention, which are very necessary for IoT devices. Furthermore, the system of Tor is actively maintained and researched. Thus, it is likely that flaws and vulnerabilities in the system can be fixed in reasonable time.

Chapter 2

Related work

In this chapter we present prior research that related to IoT security. We present the previous approaches and discuss about the difference between our approach with previous ones.

We also review previous research that related to IoT anonymity. There have been a few papers that attempts to combine the Tor network with IoT systems to achieve anonymity, but the problem has not been studied a lot.

2.1 Papers on IoT security

Privacy-preserving architecture for IoT devices has been studied widely. Song *et al.* [16] design a privacy-preserving communication protocol for IoT applications. Their protocol enables IoT appliances and sensors to securely communicate with a central controller and ensures data integrity and authentication by incorporating MAC (Message Authentication Code) to the data transmission. In contraction, our approach uses onion network to achieve secure communication.

Fabian *et al.* [9] discuss about a privacy-preserving P2P data infrastructure for IoT devices based on the Octopus distributed hash table [17] and measured the efficiency of their infrastructure (latency) using simulated network. While our design does not cover storage issues (the data are only stored locally and stashed immediately after served to the user), it is possible to extend our design by adapting their solution.

Apthorpe *et al.* [5] review and list the 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 adapt the mechanism of traffic cover to prevent analysis on traffic.

Alrawi *et al.* [3] give 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. In this thesis, we propose a privacy-preserving framework for home IoT devices by introducing the Tor network to the communication.

2.2 Papers on anonymity

Moving towards anonymous communication for IoT systems, there have been approaches utilizing onion networks. Hiller *et al.* [10] propose a mechanism to tailor onion routing to the IoT by bridging the protocol incompatibilities securely offloading expensive computations to an external server owned by the IoT device owner. Their work focus on solving problems of incompatible protocols and constrained resources. While our approach propose a framework which is designed to communicate through Tor, the optimization is also adaptable in our approach to provide a more generalized design for privacy-preserving IoT systems.

Hoang *et al.* [11] discusses about the challenges and benefits of using Tor network to secure smart home appliances. They list a number of vulnerabilities that IoT users are facing, and show how Tor-based communication can help users protect their privacy. Contrary, we present a concrete design and implementation for a specific type of IoT device (an Internet-enabled video doorbell).

Focusing about enhanced security communication for IoT addressing and connectivity, Baumann *et al.* [6] discussed about how utilizing Tor network benefits in environments behind firewalls, proxies and NAT. They have also proposed a prototypical implementation for a Internet-enabled 3D printer using a RaspberryPi as the hardware. Our approach provides an implementation for a different type of IoT device and includes more security and usability features.

Chapter 3

A privacy-preserving framework for IoT devices

In this chapter, we propose a privacy-preserving framework for generalized IoT devices by utilizing the Tor network. 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 interacts 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 the discussion section.

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 same to an ISP.

Traditionally, IoT devices run globally searchable services[4]. Consequently, the IoT devices are exposed to potential vulnerabilities and attacks. Attackers can exploit

vulnerabilities by measures such as enumeration or DoS attacks, in order to steal the data or takeover the whole device.

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

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

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 are able to eavesdrop, analyze modify the user's traffic in aggregate.

Furthermore, the adversary can obtain and analyze IoT devices and client devices. They may inspect into 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, modify the packets or send packets to the user's phone at his will.

3.2.2 Security Guarantees

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

- **Anonymity on both side**: The communication between the client and home IoT device should behind Tor. *Out-of-network adversaries* should get identical information (including IP address and physical address) of neither the client or 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* or *Out-of-network adversaries*

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

3.3 Feature

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

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

- **NAT-piercing.** For compatibility with typical home networks, the system should work on devices that are located behind firewalls or gateway routers and do not have a public IP address.

- **Decentralization.** The system should try to avoid all points of decentralization. The user should not need to register to third-party services or external websites to have the service running or retrieve data.

- **Real-time data transmission.** The system should let its user to retrieve data instantly, and send notification to end-user device whenever the sensor has something detected.

3.4 Methodology

In order to achieve the security guarantees described above, we make the approach to introduce 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:

- Anonymous connection: Taking advantage of the structure of onion network, Tor client users are kept anonymous. Traffic packets are sent across multiple volunteer-hosted nodes (called onion relays). Packets are altered at each of the nodes by either adding or removing one layer of encryption. The client establishes a circuit, involving a number of onion relays to the destination. In a Tor circuit, each relay only knows 1) which relay it is sending data to and 2) which relay has sent it data. In such structure, the client user is kept anonymous during data transmission.

- Hiding service locations: In addition, onion services also keep the server (receiver) anonymous. By the nature of anonymity, it protects the server from denial-of-service (DoS) and physical attacks. As a side effect, onion services can run on devices that are behind a firewall or gateway router, and

thus providing the feature of NAT-piercing. Given that most home network are behind gateway routers and home IoT devices usually do not have a public IP address, such side effect is very attractive for IoT devices.

Chapter 4

Case-study: A privacy-Preserving Video Doorbell

In this chapter, we present a case study which focuses on the design and implementation of a specific type of IoT devices - an Internet-enabled video doorbell. We assume the communication model to be the user-to-IoT model, where the IoT device (video doorbell) is located in the user's home network and communicates with the end-user device (smartphone).

4.1 System design

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

4.1.1 Use case analysis

We consider there should be four main 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 he wishes.
- **Face detection and push notification.**Whenever a visitor shows in front of the camera and/or pressed the physical doorbell button, all users registered to the doorbell should receive a push notification message.

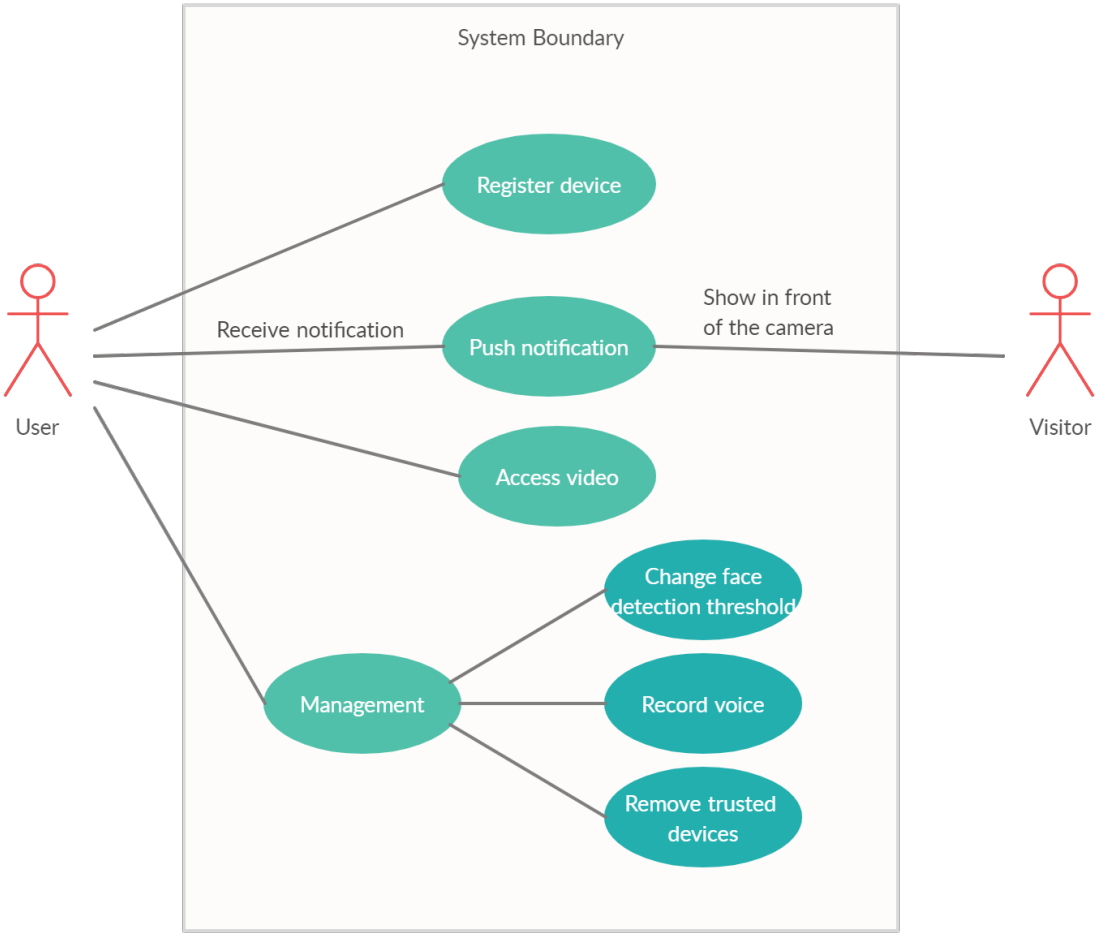

Figure 4.1

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

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

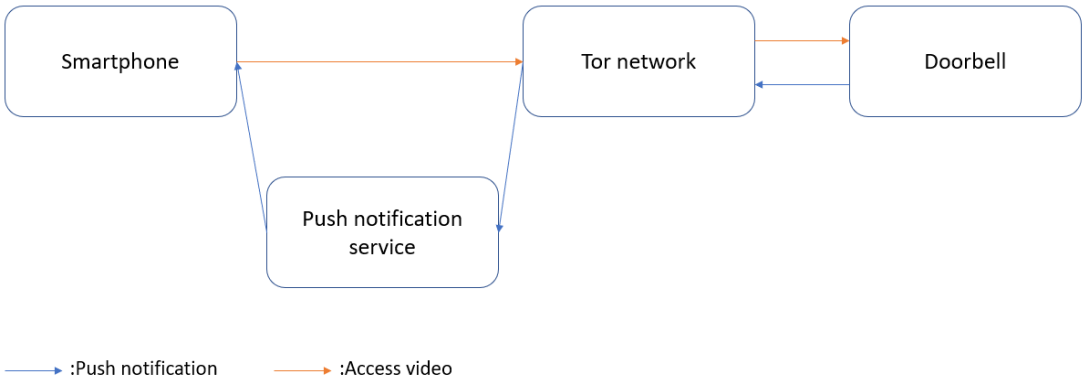

Figure 4.2

4.1.2 Network architecture

Fig. 4.2 shows the network architecture of the system. Both the client and server are behind Tor for anonymity. In addition, we configure Tor on the doorbell side to use Snowflake [1] pluggable transport for obscuration.

4.1.3 Flow structure

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 easily register (by searching the doorbell using mDNS). During the registration process, the app performs a key exchange with the doorbell and receives randomly generated credentials for user to set up in Tor (Orbot).

Playing video and authentication. As soon as the user presses the PLAY button in the app, it will work with Orbot and send a RTMP PLAY request to the

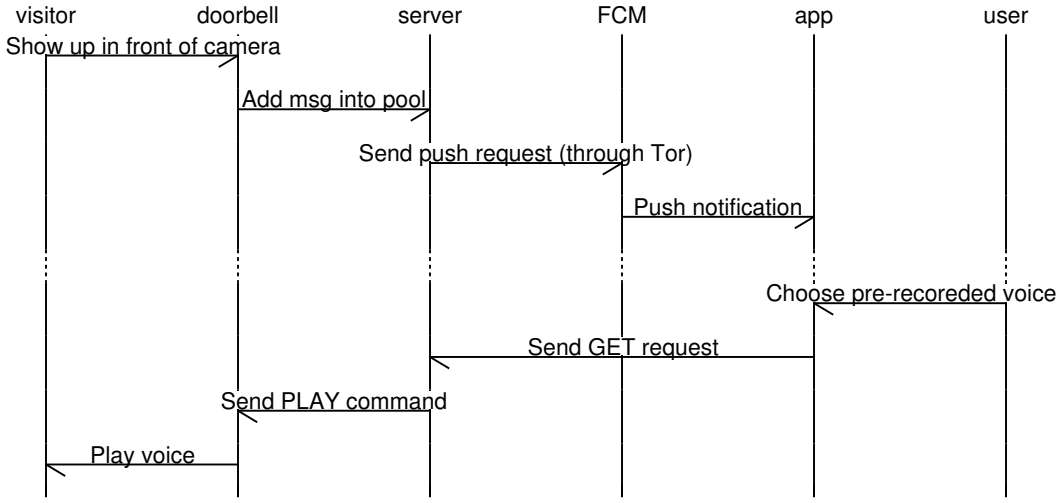

Figure 4.3

video server running on the doorbell. The video server will then forward an authentication request (in the form of a 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. 4.4* shows the flow of playing video and authentication.

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 a visitor appears, the system should send a notification to the user's devices.

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. 4.5* shows the flow of face detection, push notification and answering the door.

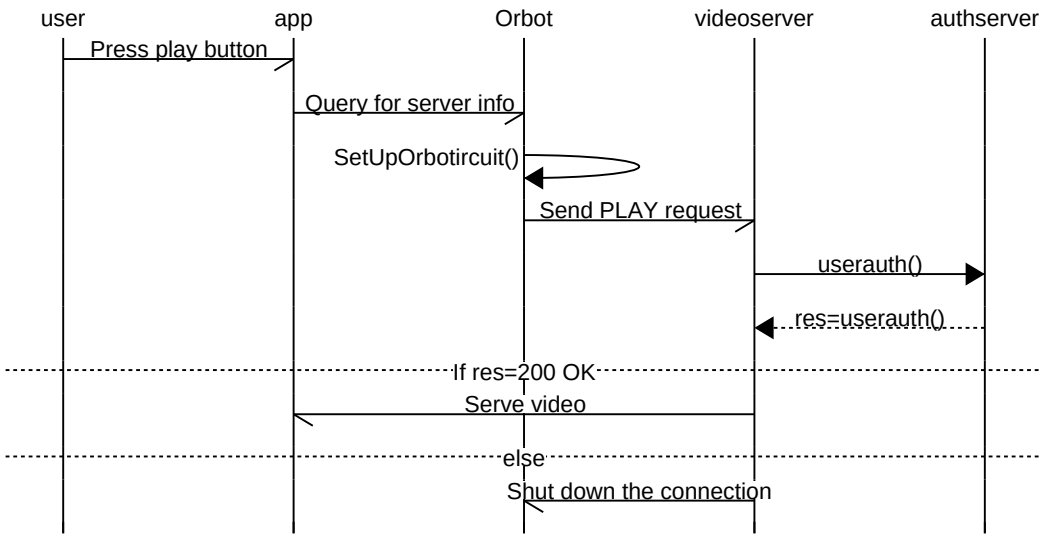

Figure 4.4

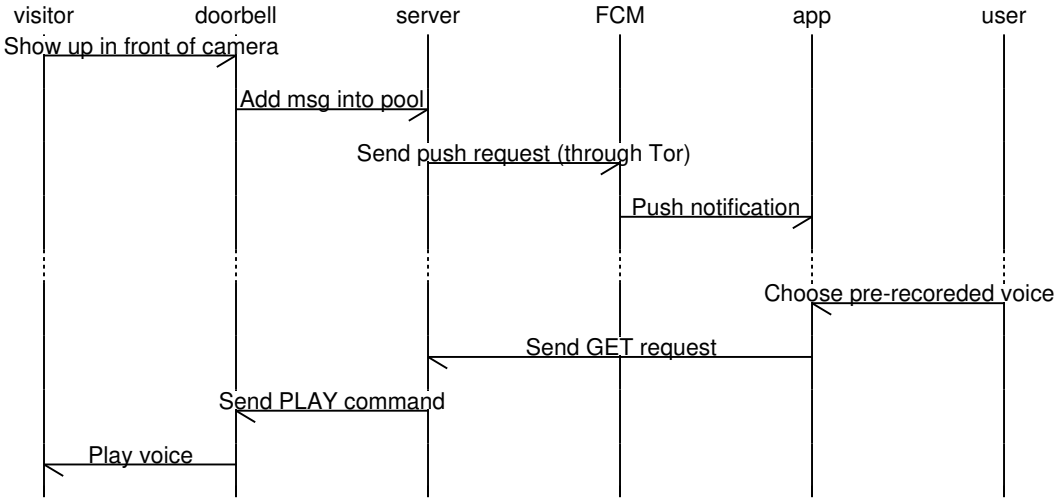

Figure 4.5

4.2 Specs of the devices

We configured a RaspberryPi 4 model B as the doorbell device. The RaspberryPi 4 has a built-in WiFi 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 more than 2GB of RAM and a CPU with 1.5 GHz, which makes it more than enough for a typical home IoT device.

4.3 Face detection

For the face detection task, we use pre-trained CascadeClassifier. Cascade Classifiers based on Haar-like features was introduced in 2001 and still widely used in face detection [15]. Pre-trained Cascade Classifiers have short execution time and small calculation load, which makes them a good choice for on-device calculations on IoT devices with low computational power.

In the actual implementation, we sample 6 frames per second, 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 interval of push notification) after a successful detection to prevent abusing resources.

4.4 Push notification

To notify the user whenever a visitor is present in front of the camera, the system adapts Google's Firebase Messaging Service.

We designed two types of messages: **BELL** (sent when someone pushes the physical button of the doorbell device) 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 from obtaining information by analyzing the 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 structure with the normal ones, but has a different type to be recognized by the client app. By padding the dummy packets to the traffic, the system sends a packet every 5 second.

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 decode the videos in flv format because it is consistent among different operating systems and good for future enhancement of support on other platforms.

Our system adapts two layers of authentication on top of video streaming. The first layer of authentication is the *onion authentication*. Provided along with Tor, it requires the user to have certain credentials set up in their Tor (Orbot) client. The onion host refuses to connect if one do not have such settings, or have different settings. The second layer of authentication is the *RTMP authentication*. In order to access the video, the user will have to add a couple of arguments in additional to their RTMP PLAY request. *Fig. 4.6* shows the components of the video serving url. In the url *appname* and *streamname* are Nginx settings and of the user's choice. *usertoken*

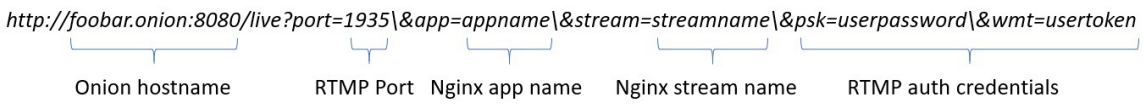

Figure 4.6

is a random number securely generated from the client app (upon first launch) and *userpassword* is calculated by the following equation:

$$userpassword = HMAC - SHA256(seed, usertoken)$$

The server checks if *all* credentials match, and shut down the connection if any are incorrect.

4.6 Android App

We implemented an Android App to pair with the video doorbell service. The app runs on the end-user device, and grant user direct 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 a HTTP POST request to port 8080 of the server (running on the doorbell device). The server will then response 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 easily set up the service.

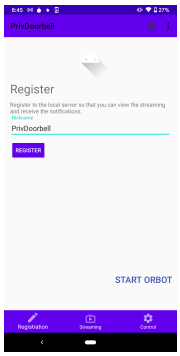

Figure 4.7

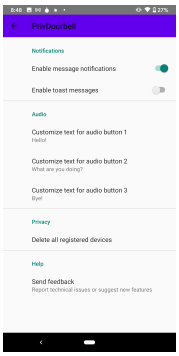

Figure 4.8

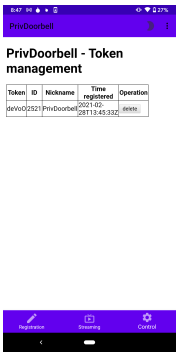

Figure 4.9

Fig. 4.7 shows the main (registration) GUI of the app.

4.6.2 Media player

The app has VLC player integrated for video streaming. Working with Orbot, the media player plays the video through the Tor network, which prevents adversaries from eavesdrop into the transmitted video.

4.6.3 Management

The client sets up a webpage for the user to manipulate settings, including changing detection threshold, revoking authentications, recording voice, etc. The management page is only accessible in local network (that saying, the user device must be in the same wireless network with the doorbell device in order to manipulate settings) and require a password for accessing.

Fig. 4.8 shows the preference page, and *Fig. 4.9* shows the token management (where user can revoke trusted devices) of the app.

4.7 Experimental study

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

- Streaming latency
- Energy consumption
- Bandwidth
- Notification latency
- The cost of using Tor

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

4.7.1 Streaming latency

We measure the latency of our system as the time difference between the user's pressing on the Play button and the first frame's appearing. *Fig. 4.10* shows the time taken for 100 samples. We divide the time into four phrases.

- Circuit: The *Circuit* phrase is from when the app starts processing the user's request to when it receives the information of the server. In this phrase, the app queries the proxy (Orbot) about the onion url. Orbot will then try to establish a circuit connection the user device and the server. In practice, the time taken in this phrase varies, as sometimes there is an existing circuit. If Tor has to create a new circuit, this phrase typically takes a few more seconds. In average, this phrase takes 2.355 seconds with a standard derivation of 3.07 seconds.
- Transmission: The *Transmission* phrase 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 a HTTP answer code 200 if the client provides correct credentials. The video transmission will begin immediately after the HTTP response. In average, this phrase takes 0.599 seconds with a standard derivation of 0.089 seconds.

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

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

In average, the whole process takes 6.867 seconds with a standard derivation of 3.141 seconds. *Fig. 4.11* shows how much time each phrase takes.

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.

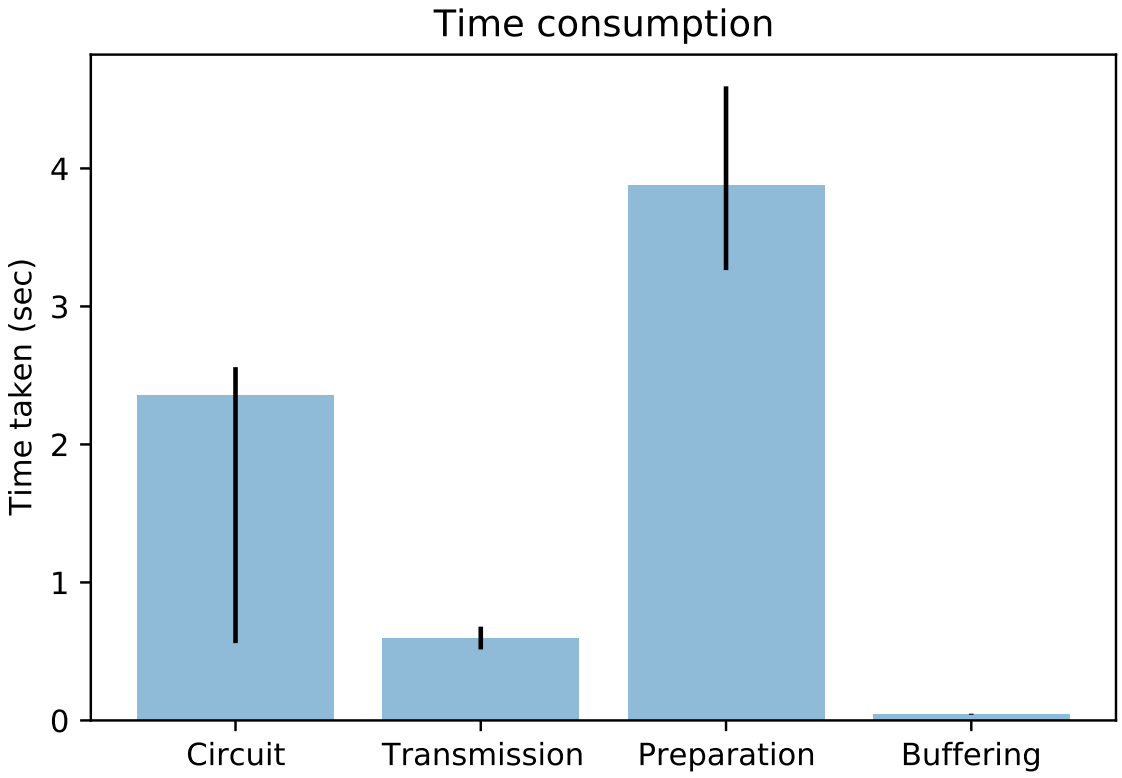
Time consumption
Time taken (sec)
0
1
2
3
4
Circuit
Transmission
Preparation
Buffering

Figure 4.10

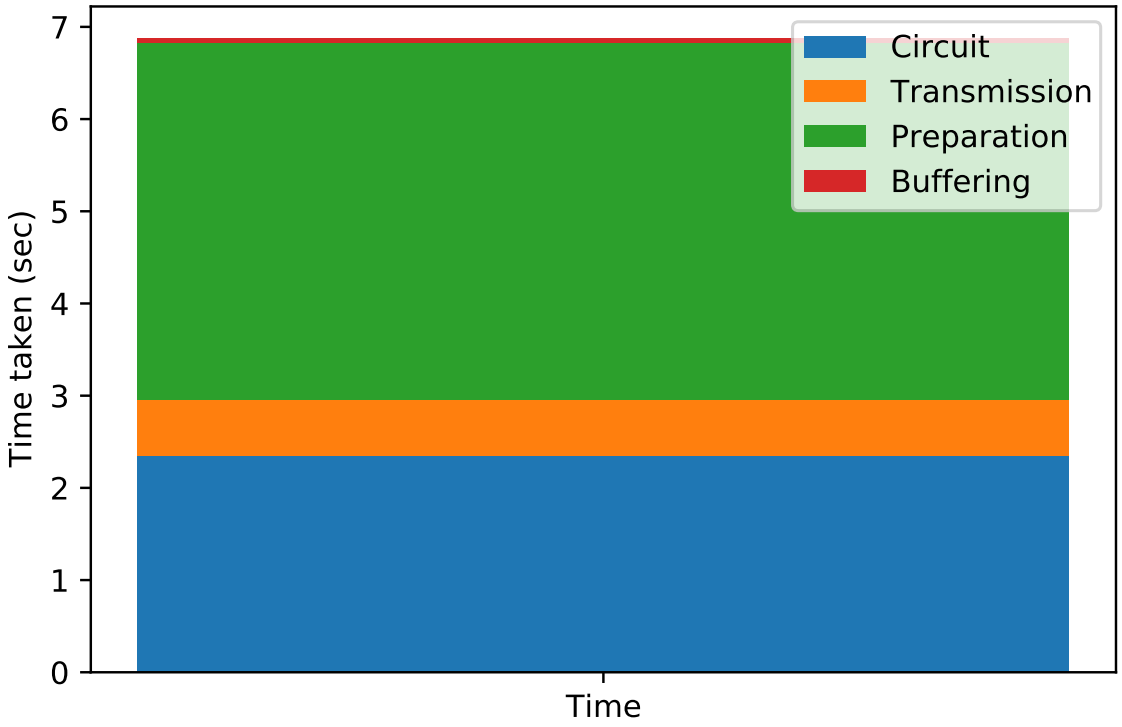
Circuit
Transmission
Preparation
Buffering
Time taken (sec)
0
1
2
3
4
5
6
7
Time

Figure 4.11

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 size of 410 kb/s.

4.7.4 Video and audio quality

The system supports videos up to 1024x768/6fps and audio with 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 second. Fig. [**?**] shows the CDF (cumulative distribution function) vs. latency (in second) diagram.

4.7.6 The cost of using Tor

Finally, we would like to 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:

- Streaming latency
- Notification latency

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

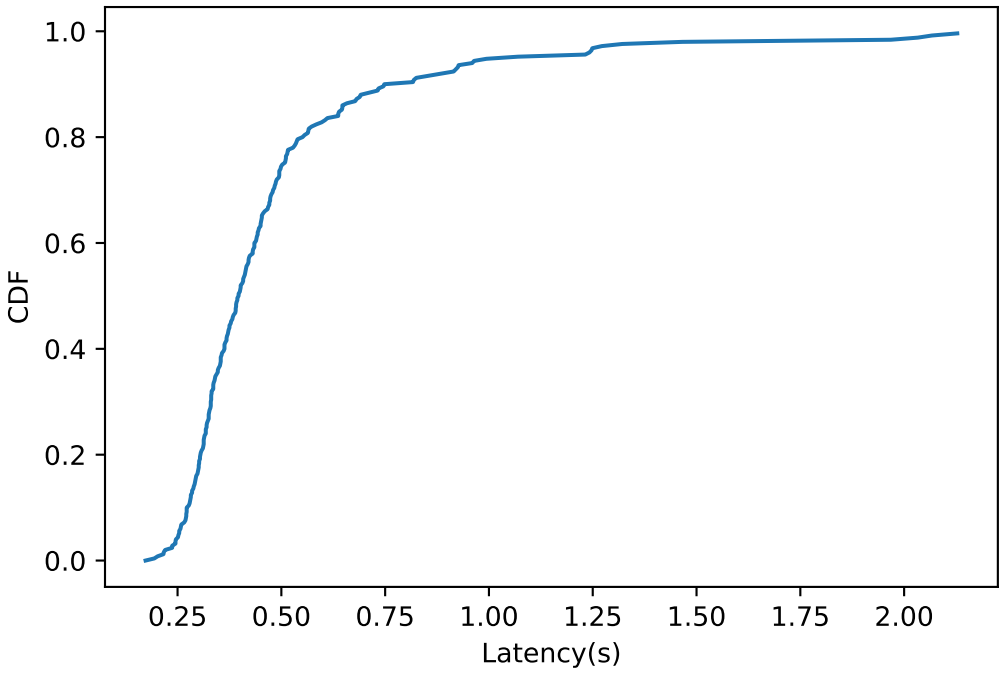

Figure 4.12

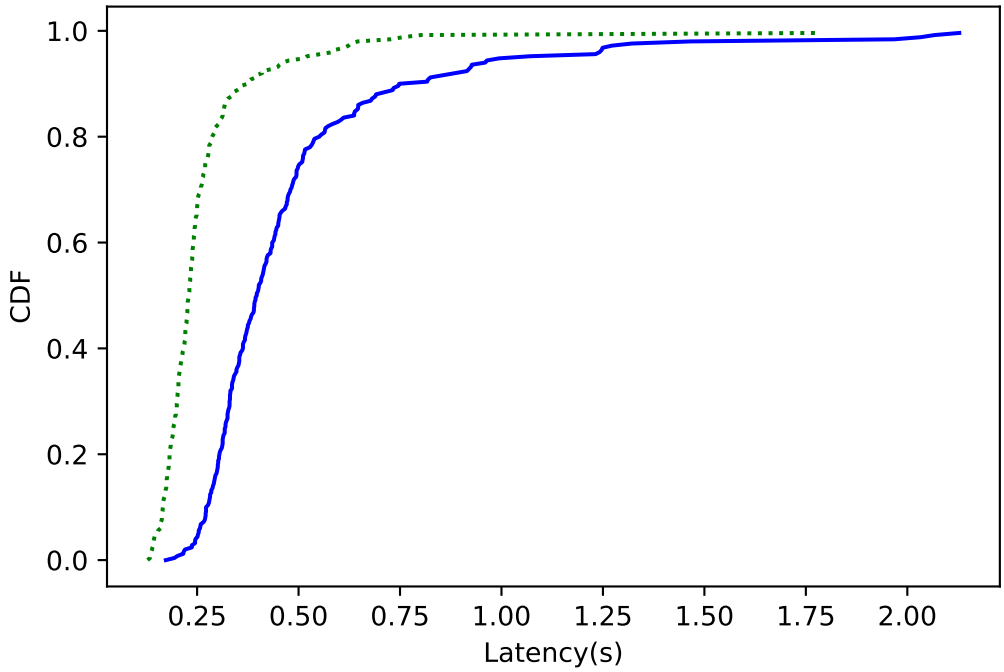

Figure 4.13

Notification latency As we described in 4.7.5, the latency of push notification with Tor is 0.478 second on average. We further conduct experiment where Tor is not in the middle. *Fig. 4.13* shows the difference of the service with Tor ("with-Tor") between that without Tor ("vanilla"). The dotted green line stands for the latency of vanilla service, and the blue full line stands for the latency of with-Tor service. The median of with-Tor service is roughly twice as the median of vanilla service.

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 supposed to be a safe countermeasure against traffic analysis, the scheme may consume much more power than necessary since the interval of people visiting is usually far larger than 5 seconds.

There have been researches discussing about efficient ways of padding traffic. For example, Juarez *et al.* [12] 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 any point of centralization and so does the push notification part. However, it is very hard to achieve fully decentralization in modern mobile systems. It is nearly impossible to push a notification with out using *APNS (Apple Push Notification Service)* on an iOS device. It is somehow more realistic on Android devices as Google permits the use of third-party push notification services. Unfortunately, most 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 the 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.* [13] 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. 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, it shows the possibility to deploy a fully decentralized push notification service with our system, and we consider the full implementation to be one of the future enhancements.

5.3 Storage

In our implementation, we did not include storage of 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 for service fee). In this case, 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. 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 [9] [14].

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 (#TODO: probably more).

5.4 IoT devices in untrusted network

In this thesis, we consider the home IoT device to be located in trusted network (i.e. home network). However, there are scenarios that such devices will also be located in untrusted network, for example the user is carrying the device on travel. In such cases, additional attack surface 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 the model is not covered in this thesis, it is possible to extend our solution in such model so that each IoT devices host its own onion service.

Chapter 6

Conclusion

This thesis first explores the challenge of designing privacy-preserving systems for home IoT devices. It reviews currently available home IoT devices, their privacy threat and the need of 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 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 prove the implemented system's usability.

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

Appendix A

First Appendix

Appendix B

Second Appendix

Bibliography

[1] [n. d.]. keroserene/snowflake: WebRTC Pluggable Transport. `https://github.com/keroserene/snowflake`

[2] [n. d.]. The Tor Project: Privacy Freedom Online. `https://www.torproject.org/about/history/`

[3] 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.

[4] 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.

[5] 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).

[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] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.