# Security and Privacy in the Internet of Things

By

Lei Yang

Submitted to the Department of Electrical Engineering and Computer Science and the
Graduate Faculty of the University of Kansas
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Committee members

Prof. Fengjun Li, Chairperson

Prof. Jun Huan

Prof. Prasad Kulkarni

Prof. James P.G. Sterbenz

Prof. Yong Zeng

Date defended: July 25, 2017

The Dissertation Committee for Lei Yang certifies
that this is the approved version of the following dissertation :

Security and Privacy in the Internet of Things

Prof. Fengjun Li, Chairperson

Date approved: July 25, 2017

# Abstract

The Internet of Things (IoT) is an emerging paradigm that seamlessly integrates electronic devices with sensing and computing capability into the Internet to achieve intelligent processing and optimized controlling. In a connected world built through IoT, where interconnected devices are extending to every facet of our lives, including our homes, offices, utility infrastructures and even our bodies, we are able to do things in a way that we never before imagined. However, as IoT redefines the possibilities in environment, society and economy, creating tremendous benefits, significant security and privacy concerns arise such as personal information confidentiality, and secure communication and computation. Theoretically, when everything is connected, everything is at risk. The ubiquity of connected things gives adversaries more attack vectors and more possibilities, and thus more catastrophic consequences by cybercrimes. Therefore, it is very critical to move fast to address these rising security and privacy concerns in IoT systems before severe disasters happen. In this dissertation, we mainly address the challenges in two domains: (1) how to protect IoT devices against cyberattacks; (2) how to protect sensitive data during storage, dissemination and utilization for IoT applications.

In the first part, we present how to leverage anonymous communication techniques, particularly Tor, to protect the security of IoT devices. We first propose two schemes to enhance the security of smart home by integrating Tor hidden services into IoT gateway for users with performance preference. Then, we propose a multipath-routing based architecture for Tor hidden services to enhance its resistance against traffic analysis attacks, and thus improving the protection for smart home users who desire very

strong security but care less about performance.

In the second part of this dissertation, we explore the solutions to protect the data for IoT applications. First, we present a reliable, searchable and privacy-preserving e-healthcare system, which takes advantage of emerging cloud storage and IoT infrastructure and enables healthcare service providers (HSPs) to realize remote patient monitoring in a secure and regulatory compliant manner. Then, we turn our attention to the data analysis in IoT applications, which is one of the core components of IoT applications. We propose a cloud-assisted, privacy-preserving machine learning classification scheme over encrypted data for IoT devices. Our scheme is based on a three-party model coupled with a two-stage decryption Paillier-based cryptosystem, which allows a cloud server to interact with machine learning service providers (MLSPs) and conduct computation intensive classification on behalf of the resourced-constrained IoT devices in a privacy-preserving manner. Finally, we explore the problem of privacy-preserving targeted broadcast in IoT, and propose two multi-cloud-based outsourced-ABE (attribute-based encryption) schemes. They enable the receivers to partially outsource the computationally expensive decryption operations to the clouds, while preventing attributes from being disclosed.

# Acknowledgements

In this section, I would like to express my gratitude to my advisor, my colleagues, my committee members and my family for their guidance, encouragement and support along the road of my PhD study.

First of all, I would like to express my deepest gratitude to my advisor, Prof. Fengjun Li, for her tremendous support, guidance, encouragement, patience and contributions to the ideas in this dissertation. I could not have done this without her help. When I began my PhD, she led me to security field and help me grow from an undergraduate to a PhD. I will never forget her patience on guiding me to identify research questions, tackle them and write a research paper. Her patience, enthusiasm, knowledge and insights have been a great source of inspiration and will also benefit my whole life. I will always cherish the time we spent working together.

I am also very gratefully thankful to my committee members: Dr. Luke Huan, Dr. Prasad Kulkarni, Dr. James P.G. Sterbenz and Dr. Yong Zeng. They offer me professional suggestions on my proposal and dissertation, which help greatly improve the quality of my dissertation. Without their help, I cannot finish the entire course of my PhD.

I would like to thank all my colleagues in University of Kansas. Dr. Wenrong Zeng, Dr. Yuhao Yang, Dr. Meeyoung Park, Yingying Ma, Abdulmalik Humayed, Hao Xue, Qiaozhi Wang and Chris Seasholtz, they have been like a big family to me and an essential part of this experience. I would like to thank them all for the fun moments we had together. I am always grateful meeting them in the painful yet rewardful PhD study.

Last but not least, I want to thank my family. My parents give me endless love and support throughout my life. Even though they know nothing about computer science, they offered me much advice and encouragement that was a great source of comfort. Finally, to my beloved wife: I am super lucky and grateful to have her in my life. She is the most beautiful, supportive, patient and humorous girl in my life. Her great support and encouragement have brought happiness and healed pain in the most important stage of my life. Without your love I would never have reached this point.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Internet of Things

The Internet of Things (IoT) is a term coined in 1999 by Kevin Ashton, a British technology pioneer working on radio-frequency identification (RFID), to describe a system where the physical world is connected to the Internet via ubiquitous sensors. Since then, although IoT has become a very hot topic in industry and academia, there is still no standard universally-accepted model for the IoT [115]. For example, the definition for high complexity IoT system from IEEE is:

"*Internet of Things envisions a self-configuring, adaptive, complex network that interconnects 'things' to the Internet through the use of standard communication protocols. The things offer services, with or without human intervention, through the exploitation of unique identification, data capture and communication, and actuation capability. The service is exploited through the use of intelligent interfaces and is made available anywhere, anytime, and for anything taking security into consideration*" [2].

The International Telecommunication Union (ITU) focuses more on the communication aspect of IoT and defines it as:

"*A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and*

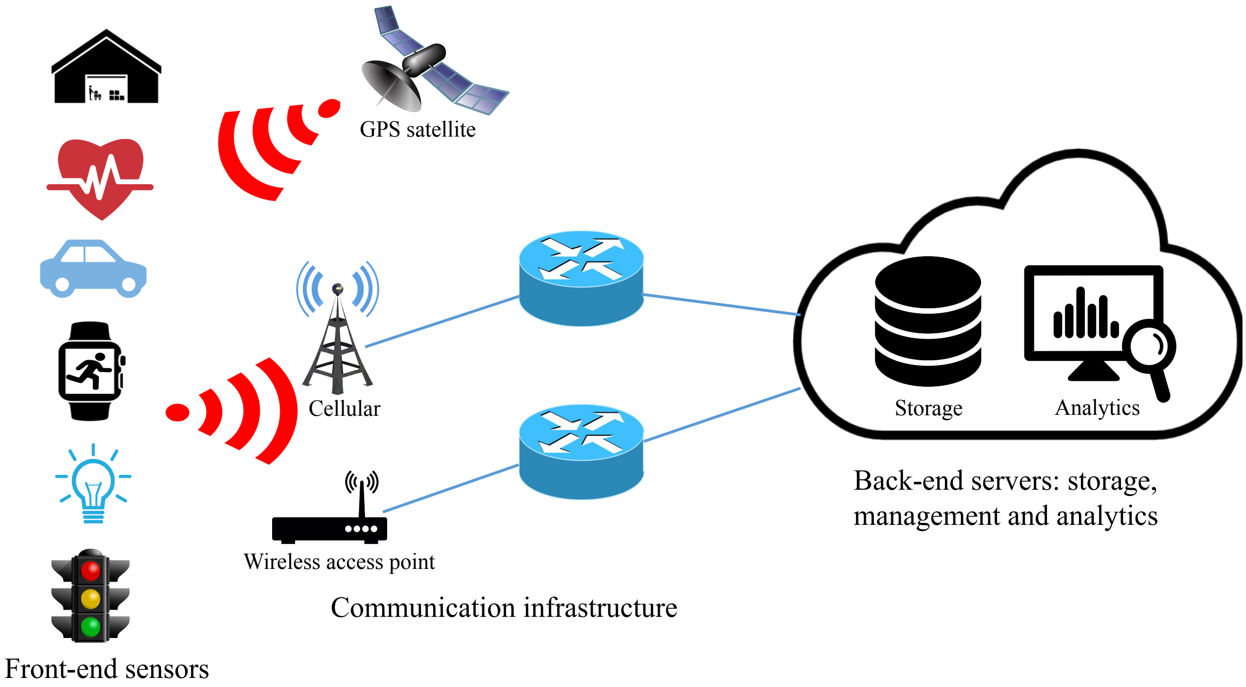

GPS satellite
Cellular
Wireless access point
Communication infrastructure
Storage
Analytics
Back-end servers: storage, management and analytics
Front-end sensors

Figure 1.1: IoT system

*communication technologies* [12]".

Since the concept was coined, IoT is changing the world to a type of information system, where objects embedded with sensors gather data, share them, and take action on them across a network. Today's computing technologies and ubiquitous connectivity have led to a pervasive deployment of intelligence into our daily life in areas as diverse as healthcare (e.g., wearable fitness tracking, remote patient monitoring), home automation (e.g., smart thermostat, security monitoring), smart grid (e.g., load balancing, smart pricing), smart cities (e.g., smart traffic control, distributed pollution monitoring), etc. By 2016, the total economic potential of IoT had already reached $120 billion and could reach $6.2 trillion by 2027 [128]. With more than 20 billion devices to be connected to the Internet by 2020, as forecast by Cisco and Ericsson [49], IoT materializes a vision of a future Internet that utilizes the sensing and computing capabilities of various devices to facilitate interaction of humans with the environment. This has led to the emergence of new categories of applications with impact across industries, businesses, and ultimately, end users. Simply put, IoT is the future of technology that can make our lives more convenient and efficient.

According to IEEE IoT definition [2], a typical IoT system is comprised of three core components shown in Figure 1.1: a collection of front-end devices with the capability of sensing, computation and communication, a back-end computing and storage facility which provides analytics and intelligence, and a communication infrastructure which connects front-end sensors to back-end intelligence. In IoT applications, the front-end devices vary in different functionalities and are tailored to a specific task. A majority of them are resource-constrained devices such as sensors, actuators and micro processors, which have very limited capacity in terms of storage, computation, communication and even energy for those ones using battery as the power. In contrast, the back-end servers are much more powerful. With the fast development of cloud computing, many services have been deployed on the cloud to leverage the cheaply and easily accessible storage and computing resources. The major cloud vendors such as Amazon AWS, Microsoft Azure and Google Cloud all provide an IoT suite to accelerate the IoT development where they help clients collect and send data to the cloud, make it easy to load and analyze that information, and provide the ability to manage the devices. In this dissertation, we mainly consider the scenarios where the back-end services are deployed on the cloud. Finally, the key enabler is the communication infrastructure where wired and wireless networks such as Wi-Fi, cellular, Bluetooth, Zigbee and Internet connect front-end devices to the back-end servers and each other. Note that, different from the conventional communication paradigm in industrial networks where the data usually flows from the end sensors to the control center in a unidirectional way, the two-way communication is adopted in IoT applications where the data is exchanged between sensors and servers. For example, connecting millions of sensors and smart meters to the utilities and power plants makes the aging power grid “smart”. The two-way communication capability not only facilitates utilities to collect a variety of real-time, fine-grained data (e.g., power consumption, voltage, phase angle) from smart meters, but also enables them to publish important messages (e.g., control commands, dynamic prices) to smart devices [114].

## 1.2 Security and Privacy concerns in IoT

However, as each coin has two sides, there is a downside to IoT: when everything is connected, everything is at risk. Theoretically, any Internet-connected device can be hacked. More connected devices mean more attack vectors and more possibilities for adversaries who are attracted by the tremendous economic benefits and scale of IoT systems. With enormous benefits brought by IoT, IoT devices also make organizations and individuals vulnerable. Generally speaking, the attack surfaces that cause the security and privacy issues fall into two categories:

### 1.2.1 Attacks on IoT Device Level

#### 1.2.1.1 Hardware Attacks

Since many IoT devices are not built for security functions and are exposed without physical protection, they are vulnerable to physical attacks on hardware such as de-packaging of chip, micro probing, reverse engineering, etc.

**Example 1**: JTAG and UART debugging test pins left on the device are still one of the most effective physical hardware attack vector available to an attacker, who can use these access points to steal credentials on IoT devices such as encryption keys [33]. Wurm et al. exploit UART of a smart device to change the boot parameters of the unit and extract the login information [156]. ■

#### 1.2.1.2 Cyber Attacks

Since IoT devices are Internet-connected, the major attacks come from cyberattacks. Some of the very frightening vulnerabilities have been exposed to the public or exploited by the adversaries.

**Example 2**: A wide range of popular Internet-connected baby monitors lack basic security features, making them vulnerable to even the basic hacking attempts. Those vulnerabilities could be exploited by adversaries to carry out nefarious actions such as monitoring live video, changing camera settings, authorizing other users to remotely view and control a monitor, and even talking to the babies [59, 125]. ■

**Example 3**: Security researchers have discovered a potential way to steal users' Google Calendar information from a Samsung smart fridge by performing man-in-the-middle attack, which is due to the poor implementation of SSL in their communication protocol. While the fridge implements SSL, it fails to validate SSL certificates, thereby enabling adversaries to sit between the fridge and the Google server to view user's Google Calendar [100]. ■

**Example 4**: A more recent severe attack was launched by an IoT device botnet named Mirai. Mirai is a type of malware that finds IoT devices such as remote cameras and home routers to infect and conscripts those zombie devices into a botnet [36]. The mechanism of Mirai is pretty simple: It continuously scans the IoT devices. If the target device uses the common or factory-default user name and pass word, Mirai logs in, inspects it and then spreads the malware across the network. Due to the huge number of IoT devices in the Internet, such a simple malware even infects around 1 million devices. It announced its existence with dramatic flair by DDoSing the Dyn DNS, a company that provides a significant portion of the US Internet's backbone, resulting in the inaccessibility of several popular websites such as GitHub, Twitter, Reddit, Netflix and many others. ■

We have heard more hack stories on IoT devices spanning from vehicles to wearable devices to many others, such as causing car accident [146] and switching off pacemakers [124]. The vulnerabilities on IoT devices make them become the targets of many attacks, which steals the privacy of an individual and even causes the death. What's worse, since the number of IoT devices is tremendous, they are not only victims of cyber attacks, but also becoming a very powerful weapon to crash any web service.

### 1.2.2 Attacks on IoT Application Data

Huge benefits of IoT lie more in data than things, which gives us the insights and help make the strategy. And meanwhile, it also gives the attackers possibilities to infer user's privacy from the combined data sources collected by numerous sensors. The attack surfaces appear throughout the data lifecycle including data collection, storage, utilization and dissemination. Due to the limited storage and computing capacity on IoT devices, nowadays, most IoT applications use cloud as their external storage and computing infrastructure, and thus the attack surface is extended to the cloud. Once data is outsourced to the cloud vendor, if we cannot fully trust them, those sensitive data are at risk. Even if the data looks like non-private, it may still be used by the attackers to refer user's privacy.

**Example 5**: Wang et al. found that hackers can use the data collected by the motion sensors embedded in smart watches to steal the information about what the user is typing on a regular keyboard [148]. ■

**Example 6:** Some experts worry that a smart watch user's health and fitness stats, location or buying habits could be discovered and later used against the owner - to deny a work promotion or an insurance claim or to cause any number of other problems [82]. ■

In the above examples, since those data do not directly reveal user's secret, they draw little attention from user and thus are less well protected. As a result, they may be transmitted through the Internet without encryption, stored in the cloud in an unencrypted format, or shared with third-party service providers without special processing beforehand. However, valuable privacy of a user could be mined by a malicious adversary with advanced techniques by combining side-channel knowledge. In addition, even if the cloud vendor honestly follows the agreement and regulation, the data on cloud is still at risk. Because of the high value of IoT data being stored on the cloud, those data become attractive targets for corporate hackers and industrial spies who rely on big data to make profits [71]. We have seen massive data breaches and data theft in recent years [97].

### 1.2.3 Countermeasures

Many countermeasures have been proposed to secure the IoT devices and the data collected by them, including but not limited to controlling the access to physical devices, ensuring strong authentication such as strong password or even two-factor authentication if possible, enforcing the ability to encrypt data at rest or in transit, and limiting the usage on the data and so on. Traditional security countermeasures heavily depend on cryptographic primitives (symmetric/asymmetric en-

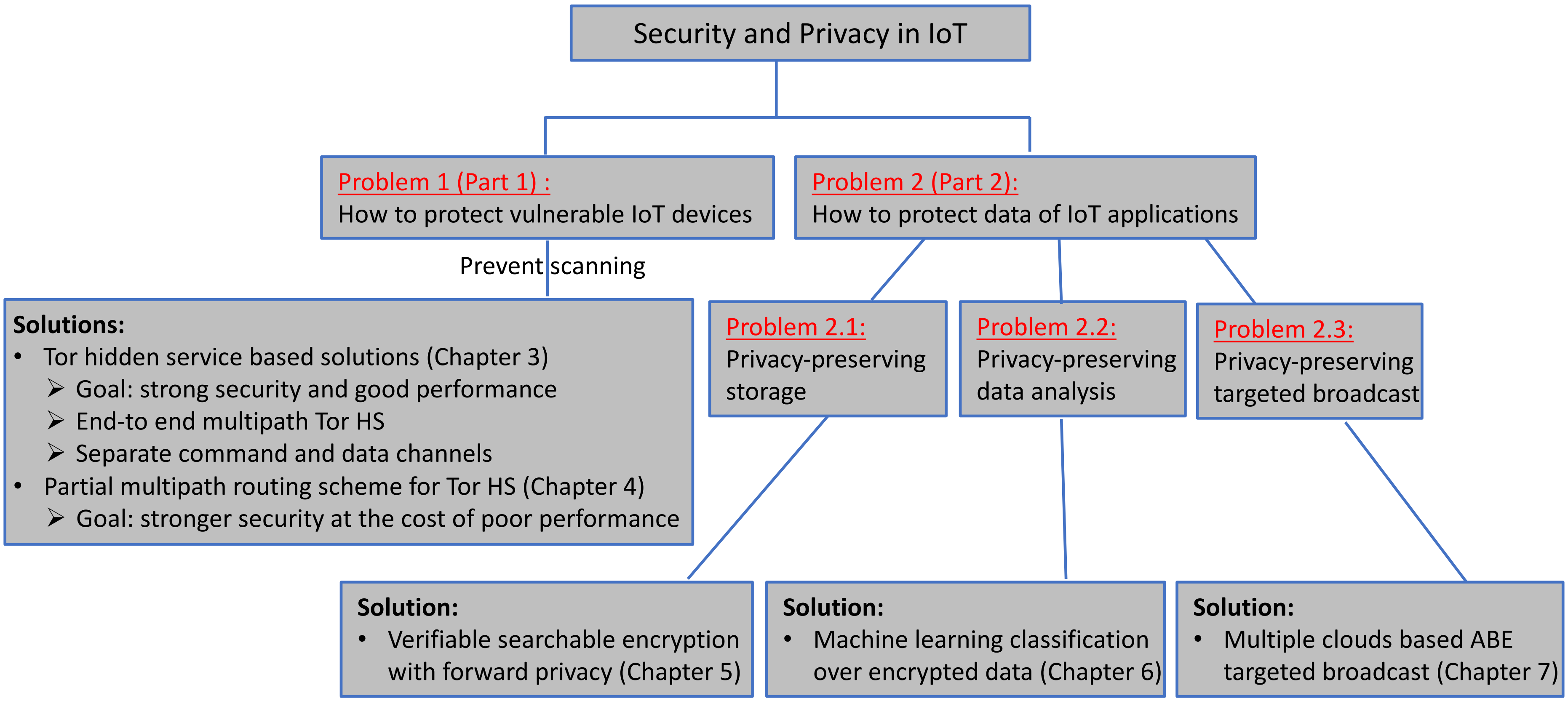

Security and Privacy in IoT
Problem 1 (Part 1) :
How to protect vulnerable IoT devices
Prevent scanning
Solutions:
• Tor hidden service based solutions (Chapter 3)
➢ Goal: strong security and good performance
➢ End-to end multipath Tor HS
➢ Separate command and data channels
• Partial multipath routing scheme for Tor HS (Chapter 4)
➢ Goal: stronger security at the cost of poor performance
Problem 2 (Part 2):
How to protect data of IoT applications
Problem 2.1:
Privacy-preserving storage
Problem 2.2:
Privacy-preserving data analysis
Problem 2.3:
Privacy-preserving targeted broadcast
Solution:
• Verifiable searchable encryption with forward privacy (Chapter 5)
Solution:
• Machine learning classification over encrypted data (Chapter 6)
Solution:
• Multiple clouds based ABE targeted broadcast (Chapter 7)

Figure 1.2: Solution overview of this dissertation

cryption, hash functions, etc.), which usually require non-negligible computing capacity, they may not be suitable for the resource-constrained IoT devices. Therefore, this raises the challenge of designing a new IoT-device-friendly approach to protect IoT device and user's data. More detailed introduction about countermeasures related to our work is presented in each chapter correspondingly.

## 1.3 Our Solutions and Contributions

The contribution of this dissertation is multifold covering both how to protect the IoT devices from cyber attacks, particularly the devices in home area network, and how to protect the privacy of users during data collection, storage, utilization and dissemination. The structure of this dissertation can be found in Figure 1.2.

The first part of this dissertation is to protect the IoT devices from cyber attacks by using anonymous communication techniques such as Tor. First, we propose two novel Tor hidden services based solutions to secure smart home: IoT gateway over multipath Tor hidden services and IoT gateway with separate command and data channels. Tor hidden service is integrated into the IoT gateway to tunnel all the data traffic from the device to the end user via Tor network, eliminating the use of the public Internet, and thus the remote users attempting to access an IoT device will have to have the Tor onion address. As a result, the hackers cannot know which devices users are connecting to, and thus, they cannot scan IoT devices that are protected by Tor, which is usually the first step of an attack. In the first solution, we use multipath routing scheme to improve the performance of Tor hidden services. With all data routed in the Tor network, this solution suits users who desire strong security but acceptable performance. To further improve the performance, our second solution transfers small-size command through Tor and transfers bulk data through the Internet so that we can leverage both the strong security of Tor and the good performance of the Internet.

Then, we enhance the resistance of Tor hidden service to traffic analysis attacks for those users who prefer stronger security than performance. In this dissertation, we propose a different multipath routing based scheme that exploits flow mixing and flow merging to distort or destroy inserted traffic patterns in a victim's flow. We believe this is an effective complement to the existing protection mechanism.

Another important contribution of this dissertation is to explore the potentials of cloud-assisted IoT security. We present a reliable, searchable and privacy-preserving e-healthcare system, which takes advantage of emerging cloud storage and IoT infrastructure and enables healthcare service providers (HSPs) to realize remote patient monitoring in a secure and regulatory compliant manner. More specifically, our system allows the data collected by patient's wearable devices to be stored on the cloud in an encrypted format, and meanwhile HSPs to search over the encrypted data without decrypting them first and verify the search results. Our system is built upon a novel dynamic searchable symmetric encryption scheme with forward privacy and delegated verifiability for periodically generated healthcare data.

Our Fourth contribution is to apply machine learning classification over encrypted IoT data stored on the cloud. In the above scheme, the user (i.e., patient) has to trust HSPs to view the raw data to analyze. It will not cause privacy concerns when the HSPs are trustworthy doctors. However, when machine-learning-as-service becomes a popular business paradigm where any organization owning data and machine learning techniques can provide data analysis services, the privacy concern arises, namely, whether we can trust them to view our data. To address this, we propose a cloud-assisted, privacy-preserving machine learning classification scheme over encrypted data for IoT devices. Our scheme is based on a three-party model coupled with a two-stage decryption Paillier-based cryptosystem, which allows a cloud server to interact with MLSPs and conduct computation intensive classification on behalf of the resourced-constrained IoT devices in a privacy-preserving manner.

At last, we pay attention to the privacy-preserving data publishing scheme for IoT applications. In this dissertation, we explore the problem of privacy-preserving targeted broadcast in IoT, and propose two multi-cloud-based outsourced-ABE (attribute-based encryption) schemes, namely the parallel-cloud ABE and the chain-cloud ABE. They enable the receivers to partially outsource the computationally expensive decryption operations to the clouds, while preventing attributes from being disclosed. The proposed solution protects three types of privacy (data, attribute and access policy) by enforcing collaboration between multiple clouds.

## 1.4 List of Related Publications

1. **Lei Yang**, Chris Seasholtz, Fengjun Li and Bo Luo. Hide Your Hackable Smart Home From Cyber Attacks. In submission. July 2017.

2. **Lei Yang**, Fengjun Li, Qingji Zheng and Xinxin Fan. Cloud-Assisted Machine Learning Classification over Encrypted Data for IoT Devices. In submission. July 2017.

3. **Lei Yang**, Qingji Zheng and Xinxin Fan. RSPP: A Reliable, Searchable and Privacy-Preserving e-Healthcare System for Cloud-Assisted Body Area Networks. IEEE International Conference on Computer Communications (INFOCOM), May 2017.

4. **Lei Yang**, Abdulmalik Humayed and Fengjun Li. A Multi-Cloud based Privacy-Preserving Data Publishing Scheme for the Internet of Things. 2016 Annual Computer Security Applications Conference (ACSAC), December 2016.

5. **Lei Yang** and Fengjun Li. Enhancing Traffic Analysis Resistance for Tor Hidden Services with Multipath Routing. the $11^{th}$ EAI International Conference on Security and Privacy in Communication Networks (SecureComm), October 2015. (**Best Paper Award Winner**)

## 1.5 Organization

The rest of the dissertation is organized as follows:

**Part 1 - Protect IoT Devices from Cyberattacks Using Tor:** Chapter 2 presents the security problem of smart home, including the challenges, case study and current countermeasures. In

Chapter 3, we present two Tor hidden services based solutions to enhance the security for smart home users who require better performance. In Chapter 4, we solve another problem of Tor hidden services, namely, Tor hidden service is vulnerable to traffic analysis. We investigates the use of multipath routing in Tor hidden services to resist the traffic analysis attacks for those users who desire stronger security.

**Part 2 - Protect Data Privacy of IoT Devices in Data Storage, Utilization and Dissemination:** This part consists of three chapters. Chapter 5 presents a reliable, searchable and privacy-preserving e-healthcare system which allows the data collected by patient's wearable devices to be stored on the cloud in an encrypted format, and meanwhile HSPs to search over the encrypted data without decrypting them first and verify the search results. In Chapter 6, we propose a cloud-assisted, privacy-preserving machine learning classification scheme over encrypted data for IoT devices. In Chapter 7, we present two multi-cloud outsourced ABE schemes for privacy-preserving targeted broadcast for IoT devices.

# Part I

# Protect IoT Devices from Cyberattacks Using Tor

# Chapter 2

# Security of IoT Devices in Smart Home: Problems and Challenges

## 2.1 Introduction

Today's computing technologies and ubiquitous connectivity are leading us into the era of Internet of Things (IoT), where a network of interconnected physical objects with the capability of sensing, computing and communication are changing the way we live and work, including our homes, offices, vehicles, cities and even our bodies. As forecast by Gartner, 8.4 billion connected things will be in use worldwide in 2017, up 31 percent from 2016, and will reach 20.4 billion by 2020. Total spending on endpoints and services will reach almost $2 trillion in 2017 [72]. Within the huge markets, smart home, interchangeably called home automation, is one of the fastest-growing IoT practices. Smart home refers to a residence that has appliances, lighting, HVAC, security and camera systems in your home network, which are capable of communicating with each other and can be controlled remotely through the Internet. Since these smart devices make the home more comfortable, secure, energy-efficient and convenient, we are witnessing an accelerating trend of smart devices flooding into our homes.

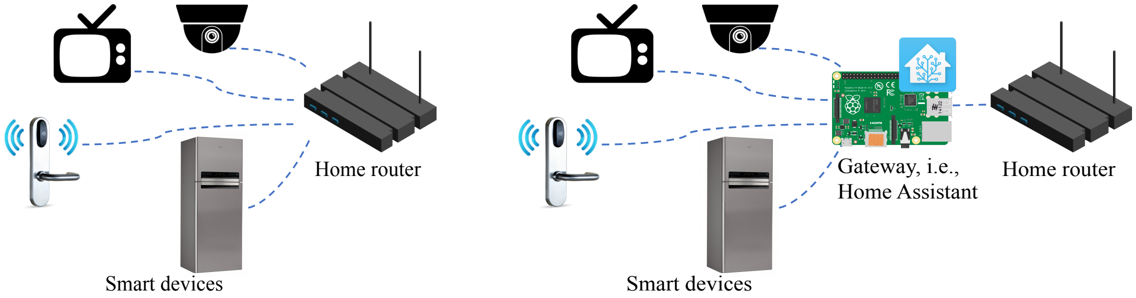
Nevertheless, when everything is connected, everything is at risk. Theoretically, any Internet-connected device can be hacked. More connected devices mean more attack vectors and more possibilities for adversaries. The threats are even more disastrous than the traditional attacks on PCs, since those smart home goods are monitoring our very private life at home. Imagine a scenario where burglars no longer have to walk around your house looking for an opportunity. Instead, they only need to hack into your surveillance system [23] or analyze your electricity metering data [162] to observe your life pattern, and finally conducts the burglary even with the help of your smart locks [53]. What is worse, hacked home devices are turned into DDoS weapons to take down the Internet. The Mirai botnet compromising millions of cameras and digital video recorders attacked the Dyn DNS servers and caused a massive network outage which may have lost companies up to $110 million in revenue and sales [46].

Consider the tremendous loss due to cyber attacks on IoT devices, security and privacy has become one of the most important aspects in the design and deployment of smart home. Various schemes have been proposed to secure IoT devices, which spans over a variety of topics including but not limited to authentication schemes [120, 105], light-weight communication protocols such as CoAP [131], RPL [152], and 6LoWPAN [130], secure operating system for IoT [25], and updating in-use IoT devices with security patches such as applying traditional encryption schemes, fixing software bugs and updating firmwares, etc. Most of the existing proposals try to find a solution that aims to secure each individual device so that we raise the fence against cyberattacks. However, they neglect the fact that all these solutions need to be implemented by device manufacturers. Even regardless of millions of insecure devices already in use, it is very difficult, if not possible, to force all manufacturers to apply security mechanisms to their new products. It is even more difficult to guarantee that the implementation is correct. These solutions can mitigate the security problem, but consider the heterogeneity of device types and even device manufacturers, the above solutions may not achieve the desired security goals in reality. Therefore, to enhance the security of smart home, if improving the resistance of each individual device against cyberattacks cannot solve the problem completely, can we find a complementary strategy that reduces the possibility of adversaries attacking on those vulnerable devices?

One possible attempt is isolation-based solution. A practical implementation is IoT gateways, which manage devices, realize home automation and integrate security mechanism. By adding an IoT gateway between home devices and the Internet, we isolate our private home from the public Internet to some extent, and thus reduce the risk of devices being exposed. Although such an isolation prevents the adversary from sniffing each susceptible device behind it, the attack surface still exists on the gateway. Therefore, to further enhance the isolation, Nathan Freitas from Tor Project [70] integrates Tor hidden services into gateway to allow users to remotely access the gateway without using IP address. By hiding IP address of the gateway, adversaries cannot scan the gateway, and thus Tor hidden services act as a buffer of security between our smart home and the adversary. The Tor-based isolation approach routes all traffic through the Tor network so that our home is never exposed to the open Internet at all. However, poor performance is a well-known problem of Tor, especially for bulk traffic such as video streaming, which is technically throttled by Tor. Users adopting this approach will experience significant delays to access the gateway through Tor network compared to direct access in the public Internet. Therefore, such an approach with disincentive of poor performance is not a practical solution in practice.

## 2.2 Smart Home Models

Figure 2.1 shows two typical smart home models. In the traditional model shown in Figure 2.1a, all home devices directly connect to the home router and allow users to remotely access each device through its own built-in interface with web browser or a mobile application. The home router is bypassed using port forwarding or remote server, e.g., cloud, which relays traffic between device and user. However, one drawback of the traditional model is that users need to install different apps for each type of device, which will increase the complexity of managing smart home. Another drawback is that this model can only provide very limited automation if the home automation is implemented through the direct interaction between devices. Image a scenario where garage door automatically opens and lights turn on when you arrive home at night. To implement this simple



(a) Smart Home model: all devices directly connect to the Internet via router

(b) Smart home model: IoT gateway manages all devices, e.g., Home Assistant on Raspberry Pi

Figure 2.1: Two smart home models

scenario, a smart garage door needs to know the sunset time and be able to inform lights of turn-on event. However, the interaction between devices are not well supported by different manufacturers because they may not design APIs for such interaction.

To solve these problems, an IoT gateway has become a viable solution shown in Figure 2.1b. The gateway can be a dedicated hardware such as a Raspberry Pi with home automation software installed. In this model, all devices are connected to an IoT gateway, which facilitates the interaction between different types of device and realize the home automation logic. Besides, instead of each device offering an interface for remote access, all devices are managed and controlled through the gateway. Nowadays, plenty of IoT gateways, such as Home Assistant [10] and openHAB [16], have been developed and used to manage various devices at home and realize home automation.

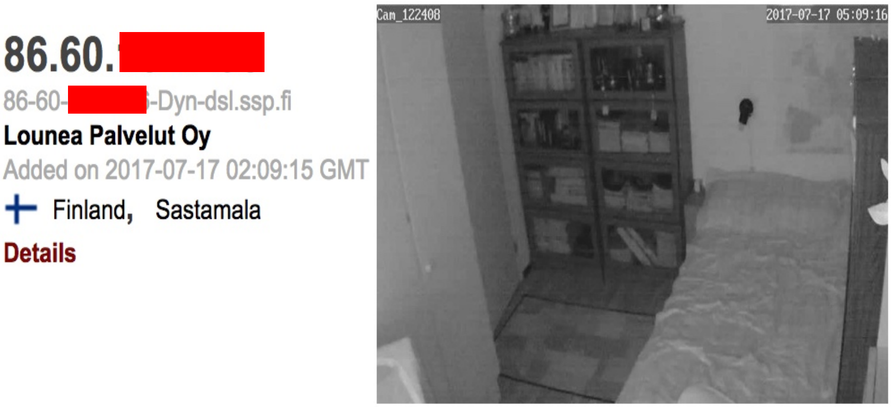
In addition, cloud-based IoT gateways for IoT are also very popular, such as Samsung's SmartThings [20], which consists of a dedicated gateway purchased by user, a cloud backend server, and a smart-phone companion app. Instead of directly accessing the gateway, users manage and implement home automation through the cloud. However, the cloud-based solutions suffer two drawbacks: (1) Because private data is stored on the cloud which cannot be fully controlled by users, users may have concerns that data is abused by cloud providers or they cannot enforce sufficient security protection for their data [66]; (2) The dedicated commercial gateway is more expensive than the Raspberry Pi with free home automation software. For example, SmartThings Hub is twice expensive than the latest Raspberry Pi 3. Therefore, huge amount of users are still using the model of IoT gateway without cloud assistance. In this paper, we particularly focus on the security problem of this model.

## 2.3 Vulnerabilities and Attacks

Simply put, a successful cyberattack on IoT device is conducted in three steps: (1) analyze the vulnerabilities of a device including hardware, system, protocol, etc.; (2) discover the vulnerable ones from billions of Internet-connected devices; (3) exploit the vulnerability.

Frightening vulnerabilities found on IoT devices at home, including baby cameras, garage doors, light bulbs, smart plugs, and more: D-LINK DCS-2132L camera is found that management interface uses plain HTTP [9]; the iBaby M6 has a web service issue that allows easy access to other people's camera details by changing the serial number in URL [1]; Ling et al. exploited the insecure communication protocols to compromise Edimax plug [108]; Samsung RF28HMELBSR smart fridge implements SSL, but it fails to validate SSL certificates, thereby enabling man-in-the-middle attacks against most connections [7]; LG 50GA6400 smart TV is hijacked by ransomware that held the display hostage [17] and also randomware for smart thermostats [8].

No one can guarantee that a device is 100% secure, and in fact, more and more emerging vulnerabilities are exposed in the market of hackers. Once the vulnerable IoT devices are connected to the Internet, hackers can easily find them using the specific search engine for IoT, like Shodan [19] and Censys [5], or their own tools. These IoT search engines are much like a network scanner, which scans the entire Internet by connecting to open ports of Internet devices and parses the headers or banners that are returned by them. The headers or banners of the reply tell us things like device type, model, vendor, firmware version and much other information [106]. The search engine attempts various protocols like HTTP, HTTPS, FTP, SSH, DNS, SIP and RTSP, etc., to scan the open ports of Internet-connected devices. Besides, IoT search engine provides API to allow other tools to access the data for legitimate purposes, but attackers can also take advantage of it. Figure 2.2 shows an example of a vulnerable surveillance camera found by Shodan by using the



86.60.[redacted]

86-60-[redacted]-Dyn-dsl.ssp.fi

Lounea Palvelut Oy

Added on 2017-07-17 02:09:15 GMT

Finland, Sastamala

Details

Figure 2.2: A screenshot of a vulnerable camera found by Shodan. We hide the last 16-bit IP address for security purpose.

search keyword “has_screenshot:true port:554”.

Finally, once these vulnerable devices are found, attackers can exploit them and launch a variety of attacks. For example, the above camera found by Shodan first leaks the privacy and then it may further lead to a potential burglary. Besides, when an attacker can compromise a huge number of devices, he may even take down the Internet using DDoS attack like Mirai [36].

**A case study**. Let us take a closer look at a recent DDoS attack, i.e., Mirai, which is a typical attack that follows the above three steps. Mirai is a botnet consisting of millions of IoT devices such as IP cameras and home routers, which caused a severe network outage in October 2016 by DDoSing DYN DNS service. The bot in Mirai does brute force scanning of IP addresses in search of other devices to infect. The bot looks for targets in an IP range and applies a port can (SYN scan) against it. If the bot successfully connects to an IP and open port, then it will try to authenticate by attempting a dictionary of known credentials or check if it can connect directly via telnet. Once the authentication is passed, the information is reported back to the command and control server, which in turn pushes down executable binary codes and uses it for DDoS attacks including UDP flooding, DNS water torture and HTTP request flooding. An infected device will repeat the above process to scan and infect other devices. Corresponding to the three steps for a cyberattack, we can see that the vulnerability is pretty straightforward - devices owners use default/weak passwords; then, infected devices discover other victims using simple port scan techniques; finally, exploitation of vulnerabilities is a classic DDoS attack. More details can be found here [34].

If we can prevent any of the above three steps, we can fail the cyberattacks on our smart home. For example, improving the security of devices and using strong password can reduce the vulnerability; protecting devices with firewall can block scanning; and filtering network traffic at ISP can mitigate DDoS attack.

## 2.4 Challenges

However, securing smart home is a challenging problem. The first challenge is that we cannot rely on device manufacturers for making cyberattack-resistant devices. They do not pay enough attention to security or even are not capable of enforcing a reliable security scheme due to several reasons: (1) In the race to be first to market and meet the need for custom-free setup, security is not their first priority; (2) To reduce the cost of cheap IoT devices such as smart switches and bulbs, small manufacturers do not apply security mechanisms on IoT devices adequately; (3) Even big vendors make mistakes when implementing security schemes, e.g., hacked Samsung smart fridge [24]. Second, since IoT devices are usually resource-constrained, they may not afford the traditional security schemes applied on PCs. Third, billions of IoT devices are already in use and many of them have fixed firmware, so it is difficult to patch security vulnerabilities for all in-use devices. Even if they can be updated, very few users will update them periodically. Fourth, the heterogeneity of devices at home makes the situation worse. Since all different types of devices from different vendors are connected at home for automation, the most susceptible device may determine the security strength of the whole smart home, and thus the adversary may eventually escalate his privilege from a cheap and vulnerable bulb to an expensive and robust lock.

## 2.5 Current Countermeasures

**Enhance the security of individual IoT device.** This type of solutions make efforts to reduce the vulnerability on device. Several dedicated communication protocols, i.e., CoAP [131], RPL [152], and 6LoWPAN [130], and operating systems, i.e., KasperskyOS [25] for IoT applications have been proposed. Since these new components have security consideration in the design, they will mitigate the vulnerabilities of IoT devices once they are widely adopted. Besides, classic security mechanisms are used in industry, such as two-factor authentication, encrypted communication, security patches and firmware update, etc. However, as we discuss in Section 2.4, it is very difficult to guarantee that every IoT device at home is secure.

**Protect all devices at home with an IoT gateway.** Corresponding to the new smart home model in Figure 2.1b, an IoT gateway with security integration has become a complementary solution for securing smart home. Note that this kind of solution is not exclusive to the above ones that enhance individual device. On the contrary, it offers one more layer protection to the smart home. In the gateway-assisted model, security mechanisms such as encryption and access control are integrated into the gateway. For example, HTTPS is used to access gateway instead of HTTP in some cheap IoT device. As long as the gateway is secure, all devices behind it is safe. Simply put, such an approach is an isolation that isolates vulnerable devices from the public Internet. However, such an isolation is not complete, which makes the gateway become the target of adversaries. They can still scan and exploit the vulnerabilities on the gateway, and thus, any security flaw of gateway will lead to protection failure. Therefore, to further enhance the isolation, Nathan Freitas from Tor Project [70] proposed an approach that integrate Tor hidden service, which is an anonymity tool that allows server operators to run a web server or SSH server, etc., without revealing their IP address to clients, into the IoT gateway. The IP address is no longer needed to access the gateway, instead, all requests and data go to a .onion address through Tor network, which is an anonymous overlay darknet on top of the public Internet, so, adversaries are not able to access/scan the gateway and any individual device any more, and even know the existence of gateway and any IoT device. However, the Tor-based isolation approach routes all traffic including bulk data such as video streaming in the Tor network, which will aggravate the performance problem of Tor. Consider the huge number of smart home users, if most of them adopt this solution, Tor will become very slow and even unavailable eventually.

# Chapter 3

# Hide Your Hackable Smart Home from Cyberattacks

## 3.1 Introduction

Using Tor hidden services to prevent adversaries from scanning the IoT gateway is a novel and promising direction to secure the smart home. However, it is well known that Tor is very slow compared to the direct access through the public Internet, which makes Tor hidden service based solution less attractive to be adopted. Therefore, in this chapter, we aim to address the performance problem of Tor hidden services. For users with strong performance-preference, to tackle the performance problem of Tor-based approach and meanwhile achieve strong security, we propose two novel isolation-based solutions: IoT gateway over end-to-end multipath Tor hidden services and IoT gateway with separate command and data channels. In the first solution, we propose a multipath Tor hidden service scheme, which constructs multiple circuits and form a tunnel between user and IoT gateway. By applying a self-adaptive scheduling scheme to transfer data over multiple circuits, our solution can achieve better performance. And meanwhile, since all traffic are still routed through the anonymous tunnel in our solution, the security level of our scheme is as strong as the original Tor-based approach. Therefore, our first solution fits users who require strong security and an reasonable performance. In our second solution, to further improve the performance but without degrading the security too much, we use separate channels for command and data transmission. More specifically, the gateway which is ran as a hidden server only accepts user's command sent through the Tor network, such as turning on video streaming for surveillance camera, which is a tiny message, while to avoid the performance disadvantage of Tor, the real data transmission for bulk traffic temporally goes through the public Internet. By separating command and data channel, we utilize both the security benefit of the anonymous Tor network and the performance benefit of the public Internet.

Besides the performance problem, Tor hidden services also suffer security problems. According to recent studies Tor hidden services are still under the risk of de-anonymization due to specialized traffic analysis attacks [121, 38]. As a result, the attacker is still able to locate the IP address of the hidden server by using traffic analysis even if it is not that easy, and thus, our Tor hidden services based solution may fail to protect the smart home. Therefore, for those users who desire very strong security but care less about performance, we propose another "partial" multipath routing scheme to improve the security of hidden services, which will be presented in next chapter.

## 3.2 Preliminaries

In this section, we will introduce some preliminaries related to our work. We will fist present the specific home automation software that we use in our work. Then, because this work is based the one type of anonymous communication techniques, namely, location hidden services, we will also give some background about the anonymous communication systems. In this work, we specifically use Tor hidden services to achieve the security goal, so we will focus on the introduction about Tor, Tor hidden services and their related problems.

### 3.2.1 One Instance of Smart Home Gateway: Home Assistant

Home Assistant (HA) is an open-source home automation platform running on Python 3, which is able to automatically discover, monitor, control and automate various devices at home [10]. It can be run on all major operating systems (Linux, Windows, OS X) and a variety of hardwares ranging from powerful computers to a cheap Raspberry Pi. Home Assistant offers a web interface and allows users to remotely access it through web browser or a mobile application. Figure 3.1 shows the interaction between components and the Home Assistant core, where the rectangles with blue background represent the Home Assistant core and the red ones are built-in components, while the green ones are the specific IoT devices. Each component is responsible for a specific domain within Home Assistant and defines the basic functionalities that all instances of this component should have. For example, light component is the interface for different brands of light bulbs, while automation component which responds to events that happen in Home Assistant is for the automation logic. In the core, Event Bus is the connection of all the components, which facilitates the firing and listening of events. State Machine keeps tracks of the states of devices and fires a *state_changed* event when a state has been changed, while Service Registry allows components to register services and listens on the event bus for *call_service* events.

We choose Home Assistant as the test platform for server reasons. First, since Freitas's Tor-based scheme is implemented on Tor [70], it is a fair comparison between his scheme and ours on the same platform. Second, since HA is a free and open-source platform with many tutorials and supports in its community, it is widely used for home automation. Third, a general-purpose Raspberry Pi with HA installed is one of the most affordable IoT gateway solutions. Although a dedicated device with hardware support such as Intel's IoT gateway [11] may provide faster processing and stronger security, it is much more expensive ($980 for an Intel gateway[4] v.s. $40 for a Pi). Fourth, our two solutions do not heavily depend on HA, which can be implemented in other platforms with a little efforts. Note that our multipath Tor hidden service-based scheme can be applied on other platforms directly.

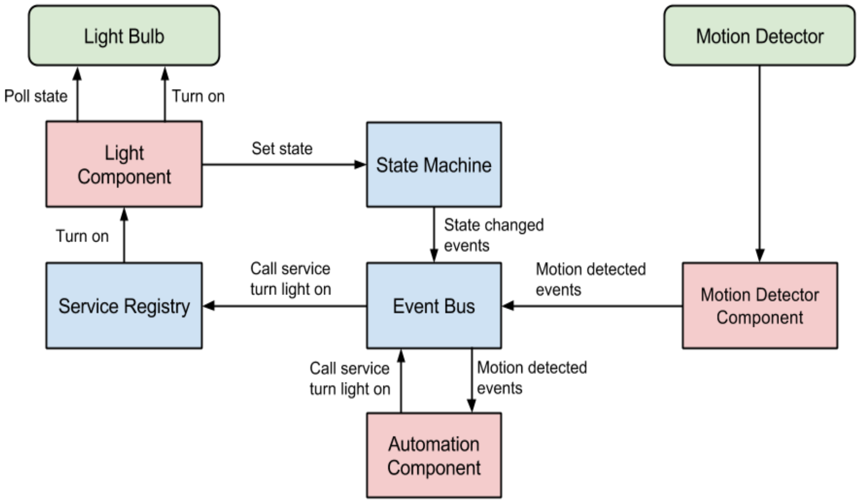


Figure 3.1: Interaction between components and the Home Assistant core [10].

## 3.2.2 Background for Tor

### 3.2.2.1 Anonymous Communication Systems

The goal of anonymous communication systems is to hide the identity of a communication participant (sender or receiver) or their linkability from being learned by adversaries. Chaum proposed the first high-latency anonymous communication system based on mix for the anonymous emails in 1981 [51]. Since then, a large number of anonymity systems have been proposed. They could be classified into two general categories: the high-latency and the low-latency systems. Generally, the high-latency anonymity systems can provide stronger anonymity due to the larger anonymity set introduced by the high latency, but they are only applicable to applications which are able to tolerate delays of several hours or even days, such as email, e-voting and broadcast messaging. Different from high-latency anonymity systems, low-latency anonymity systems usually have better performance, which are designed for real-time interactive applications, especially for web browsing, instant messaging and SSH. Since we focus on the low-latency anonymity systems in this dissertation, particularly Tor, we only discuss the low-latency anonymity systems and the details of Tor in the following sections.

#### 3.2.2.2 Low-latency Anonymous Communication Systems

Unlike high-latency anonymity systems [51, 58, 116, 54] introducing delays on the order of several hours or more, low-latency anonymity systems are designed to provide better performance to support interactive applications at the cost of vulnerability to some certain traffic analysis attacks. Various low-latency anonymity systems are proposed for commercial intention or research purposes. For example, Anonymizer.com [3] is a leading commercial anonymity system, which is a proxy accepting client's traffic and replacing the client's IP address with its own address. This approach is simple and efficient, but the centralized structure is vulnerable to subpoena attacks from intelligence agency. Distributed anonymity systems can overcome this drawback, such as Crowds [126], onion routing [76], Tarzan [69], and Tor [60]. Tarzan and Tor are both based on onion routing. In a onion routing network, clients select a sequence of proxies to build an anonymous path to the intended destination. Payload and the next-hop router in the path are encrypted in nested layers. For example, a client S sends a message M to D via three proxies A, B and C. The leaving message from S looks like $E_A(B||E_B(C||E_C(D||M)))$. After receiving this message, A peels of the outermost layer to learn that B is the next hop and forwards the remaining part to B. This process is repeated until D receives the original message. In onion routing, each relay router along the path only knows the predecessor and successor, so the anonymity is achieved in a distributed manner. Crowds uses a different approach to achieve anonymity: each client runs proxy called *jondo* locally. When a *jondo* receives a message, it will either forward the message to a random *jondo* when $x > p_f$ where $x \in [0,1]$ is randomly selected and $p_f$ is predefined, or directly deliver the message to the actual receiver. Next, we present the detailed background on Tor.

#### 3.2.2.3 Tor

Tor, also known as the Second-Generation Onion Router, is the most popular and widely deployed tool for anonymous communications over the Internet today. It provides services to millions of users on a daily basis [141] and allows users to access generic Internet services and location-hidden services. The Tor network is a distributed overlay network of more than 6,000 relays operated by

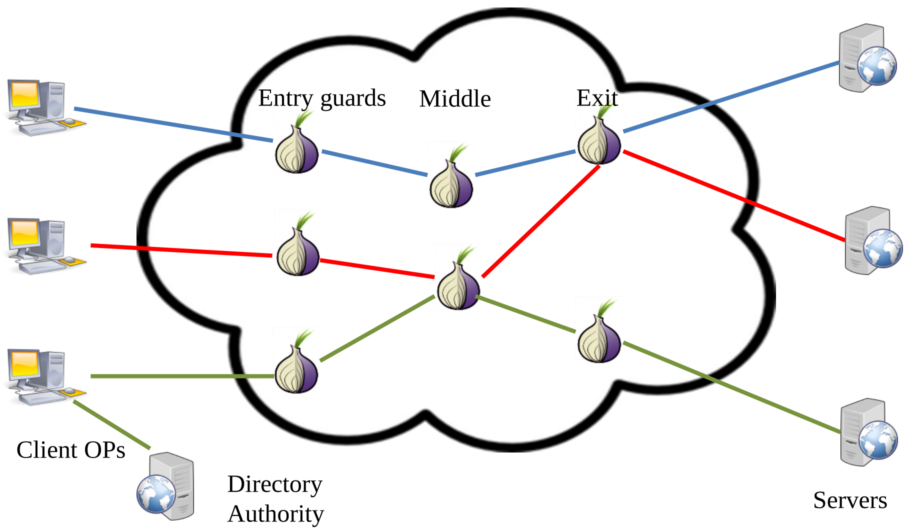


Figure 3.2: An example of the Tor Network

volunteers all around the world. These Tor relays are known as *onion routers* (ORs). Each relay operator configures his OR about its IP address, port, public key, bandwidth capacity and its exit policy. The configuration information contained in a *router descriptor* is uploaded to *directory authorities* , which also actively measure the real bandwidth capabilities of each OR, negotiate the network status with each other and publish a consensus containing the network parameters and all ORs' information. Tor clients, known as *onion proxies* (OPs), periodically download the consensus file from directories, and build virtual path, referred to *circuits*, through several available ORs which are selected in proportion to their weighted bandwidth from the consensus. Generally, a Tor circuit consists of three ORs, which are called entry guard, middle, and exit, depending on their positions in the circuit. Tor only works for TCP streams and can be used by any applications with SOCKS support. Therefore, OP attaches a client TCP stream to a circuit, encrypts the application data in layers, packs them to 512-byte *cells* and sends data cells through the circuit. Each relay along the circuit decrypts its layer and forwards the cell to the next relay until it reaches the last relay (known as "exit"), which further forwards the data to the original destination. Each hop only knows who has sent the data (predecessor) and to whom it is relaying (successor) due to layered encryption. Figure 3.2 shows an example of Tor network.

**Circuits and Streams:**

To avoid delays, an OP usually preemptively maintain a certain number of clean circuits, which are circuits that have not yet been used for any streams. When a new stream arrives, OP can attach it to a pre-established circuit quickly. If no existing circuits satisfy the exit request of the stream, OP builds a new circuit on demand. To build a default 3-hop circuit, OP selects an OR for each position in proportion to its weighted bandwidth. These weights are computed according to all ORs' real capacities, the fraction of exit and guard nodes to total bandwidth, and the position they are being selected for. OP first chooses the exit node, followed by the entry guard and middle node. The selection follows some constraints to guarantee that no same OR appears twice in a circuit, no routers come from the same /16 subnet, and all selected routers are running. After all routers are selected, OP incrementally builds a circuit through them. The circuit is extended one hop at a time, and negotiates a session key with the router through the Diffie-Hellman handshake. Each extended router only knows which router sends it data and which router it is forwarding data to. Once the circuit is established, it is ready to accept client's application streams. Usually, a circuit can be multiplexed by many TCP streams. Any TCP-based applications can use Tor via SOCKS proxy which is listened by OP. The data of each stream is padded into fixed-sized cells (i.e., 512 bytes). The data cell format is shown in Figure 3.3. Then, the onion-encrypted data cells are sent through the circuit to the intended destination.

**Flow Control:**

Tor uses a two-layer window-based end-to-end flow control scheme to guarantee a steady flow between the client and the exit. Since multiple streams multiplex a circuit, the outer layer is circuit-level control which restricts the number of cells that can be transmitted on a circuit for all streams. The inner layer is a stream-level control applied to individual streams. At two edges of a circuit, OP and exit control the speed of data cells entering and leaving the circuit by keeping track of circuit and stream windows. By default, a circuit window starts with 1000 cells (500KiB) and a stream window is initialized to 500 (250KiB) cells. When a data cell is sent, both windows will be decreased by one. When a stream window is decreased to zero, the sender stops sending from

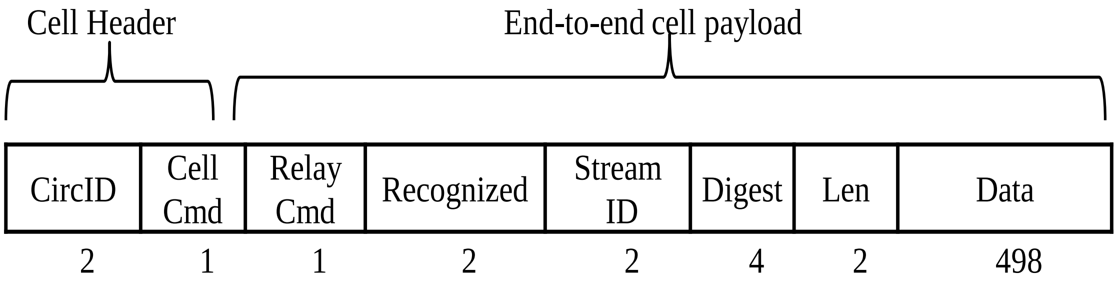


Figure 3.3: Data cell format

this stream. When a circuit window reaches zero, the sender stops sending from all streams on this circuit. Windows are increased when the corresponding acknowledgment cell known as *SENDME* is received. For every 100 cells received on a circuit, the receiver sends a circuit *SENDME* to inform the sender to forward another 100 cells from this circuit. For every 50 cells received from a stream in this circuit, the receiver sends a stream *SENDME* to request another 50 cells from this stream.

**OR-Connection Multiplexing:**

ORs in Tor connect to each other using TCP connections, in other words, each pair of onion routers has a single TCP connection. Multiple circuits that belong to different clients may share the same pair of ORs, so all the circuits between the pair are multiplexed over the single TCP connection. As a consequence, the TCP congestion control scheme will be unfairly applied to all circuits on the pair, that is, the circuit carrying low volume of traffic will be negatively affected by the circuit with large amount of traffic. In the current Tor network, it is recognized that the performance low-volume interactive traffic is greatly affected by the high-volume bulk traffic. In addition, since Tor allows OR operators to limit their bandwidth usage through a token bucket approach, the bandwidth contributed by each OR is very limited. All circuits on an OR have to contend for the limited bandwidth resource. Therefore, it is very important to reduce the influence of bulk traffic over interactive traffic. The current Tor does it through circuit scheduling. Before going to the details of scheduling, we first present the cell processing and queuing architecture inside an onion router.

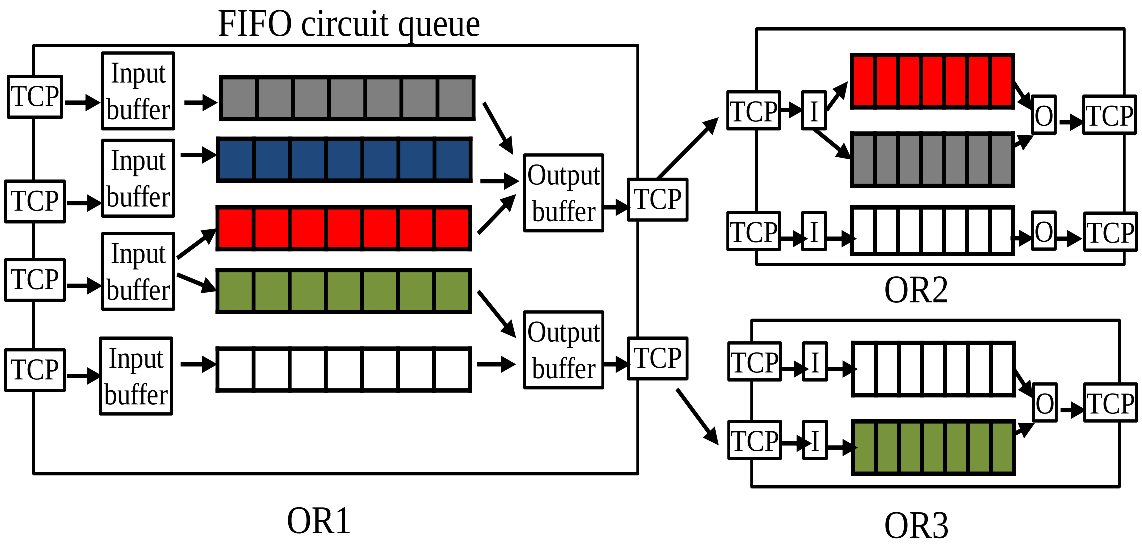


Figure 3.4: Queuing architecture inside an OR

**Cell Processing and Queuing Architecture:**

Figure 3.4 visualizes the inner queuing architecture and the connection between ORs. When a data cell arrives at an OR, the cell is first stored in the kernel TCP buffer, which is later retrieved to Tor input buffer by Tor through libevent [15]. Since an OR may connect to multiple ORs, the socket of each TCP connection is registered to libevent scheduler and associated to a notification callback function. The livevent scheduler will asynchronously executes the callback function on the active socket.

Once a data cell arrives in input buffer, it is immediately processed: the data cell is either encrypted or decrypted according to its direction. After the process, the cell is sent into the corresponding first-in-first-out (FIFO) queue based on the circuit identify (circID) in the cell. Cells stay in the circuit queue until the circuit scheduler chooses an active queue to send cells to the output buffer.

**Circuit Scheduling:**

Tor provides two circuit scheduling policies when the output buffer has space and more than one circuit have cells to send. The first one uses round-robin scheduling to ensure the fairness of bandwidth sharing among all circuits. However, due to the distinction of different applications, for example, interactive applications like web browsing have higher requirement on timeliness, round-robin scheduling may hurt the interactive traffic when a bulk traffic circuit is on the same router. Another optional policy is exponentially-weighted moving average (EWMA) scheduler, which is the default scheduler [139]. EWMA prioritizes the circuit with interactive traffic over bulk traffic through choosing the circuit with lowest packet count.

**Tor Hidden Services:**

Another significant feature that distinguishes Tor from other low-latency anonymity systems is *Tor hidden services*, which allows server operators to hide their locations (i.e., IP addresses) while providing a variety of Internet services. The Tor hidden services use rendezvous points(RPs) to support hidden TCP-based services, such as web servers and instant messaging servers, without revealing real IP addresses of hidden servers. Figure 3.5 illustrates basic components of Tor hidden services: (1) To make a service reachable, the hidden server (HS) selects several routers at random as introduction points (IPs) and builds circuits to them. IPs wait for connections on behalf of the hidden server. (2) HS then uploads its service descriptor to the hidden service directory (HSDir). The descriptor containing its public key and a set of introduction points signed by the private key. After this step, HS is ready to accept connections from clients. (3) To connect to the hidden service, we assume a client (Alice) learns about HS's onion address out of band. Then, Alice contacts HSDir and retrieves the service descriptor of HS using this onion address. (4) After getting the set of introduction points and HS's public key from the service descriptor, Alice randomly selects a router as the rendezvous point (RP) by assigning it a rendezvous cookie (RC) which is a one-time secret, and builds a circuit to it (i.e., client circuit). (5) After that, Alice sends an introduce message to one of the introduction points and (6) asks the IP to forward it to HS. The message containing the rendezvous cookie, RP address and the first part of a Diffie-Hellman (DH) handshake is encrypted by HS's public key. (7) After decrypting the introduce message, HS establishes a new circuit to Alice's RP (i.e., hidden server circuit), and sends a rendezvous cell with RC and the second part of DH handshake. (8) If RP verifies RC successfully, it then relays the rendezvous cell to Alice. After generating the end-to-end session key, Alice and HS start communicating with each other through

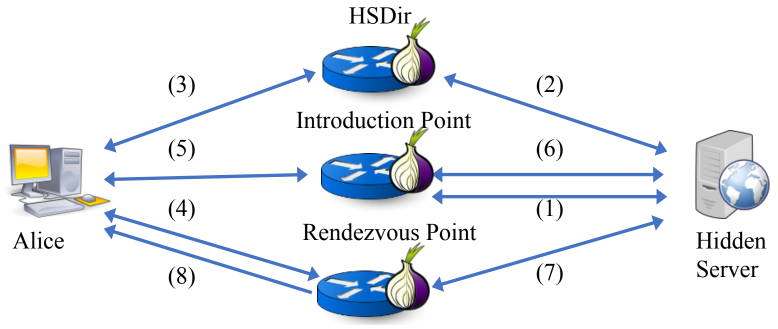


Figure 3.5: Tor hidden services architecture

RP.

## 3.3 Tor's Performance Problem

Recall that Tor was initially designed to provide anonymity service with low latency and reasonable throughput for interactive applications such as web browsing, instant message and SSH. However, Tor users often experience performance with large variance, which becomes a big obstacle to impede Tor's further expansion.

Many causes lead to Tor's performance variance. Two main reasons that slow down Tor are: (1) the path selection schemes designed for balancing traffic over the entire Tor network [133, 149], and (2) bulk traffic contending for bandwidth with interactive users [113, 139, 91]. The current path selection scheme excludes slowest relays while selecting from the remaining relays in proportion to their weighted bandwidth (i.e., consensus weight). In other words, a router with higher weighted bandwidth is more likely to be selected. The preference on high-bandwidth relays in the current path selection scheme makes some routers persist in being under-utilized or congested [149]. A single congested relay becomes the performance bottleneck of a Tor circuit between client and server. It is even worse for Tor hidden services, since 6 relays are used to connect client to hidden server. Any of them may become congested, which increases the probability of the circuit getting congested. On the other hand, bulk traffic also significantly affects the performance under the current Tor architecture. McCoy et al. found that 40% of network capacity is consumed by only a small fraction of BitTorrent users (3.5% of all connections) [113]. To achieve a good performance, most of BitTorrent traffic go through the scarce high-bandwidth relays, which increases the congestion and degrades the performance of interactive users. The performance problem may become worse in the era of IoT, which is one of the biggest sources of big data.

To solve Tor's performance problems for interactive traffic, the existing work follows three thrusts. The first attempt is to increase the overall network capacity. [89, 119, 117, 90] proposed various incentive schemes to attract more people in contributing more Tor relays. The increased overall bandwidth benefits both interactive and bulk data clients. However, we argue that without an effective allocation mechanism, the newly introduced resources may attract more bulk traffic requests that consume bandwidth more aggressively than interactive traffic and thus make the situation worse. Along the second direction are the schemes aiming at optimizing path selection based on relays' geographic locations [27] or performance history [133, 149]. A better performance is achieved for interactive users by avoiding instinctively slow relay and/or transiently congested relays being selected in the anonymity route. However, while optimizing the performance of individual circuits, these approaches reduce the global utility of the anonymity network by imposing too many restrains that excludes the capacity of low-bandwidth relays despite very limited. A different attempt in improving the performance of interactive traffic is to prioritize it over bulk traffic [139] or completely throttle bulk traffic [91]. However, one problem with these schemes is that the metrics used to detect bulk traffic are simple to game [29]. Besides, the throttling techniques enforce strict restrictions to block bulk traffic resulting in reduced overall network utilization.

## 3.4 Basic Solution: IoT Gateway over Multipath Tor Hidden Services

To overcome the performance problem of the current deployment of Tor for IoT gateway, we present an end-to-end multipath routing scheme for Tor hidden services, namely *m*TorHS. As illustrated in Figure 3.6, *m*TorHS constructs an anonymous *tunnel* consisting of *m* circuits where *m*

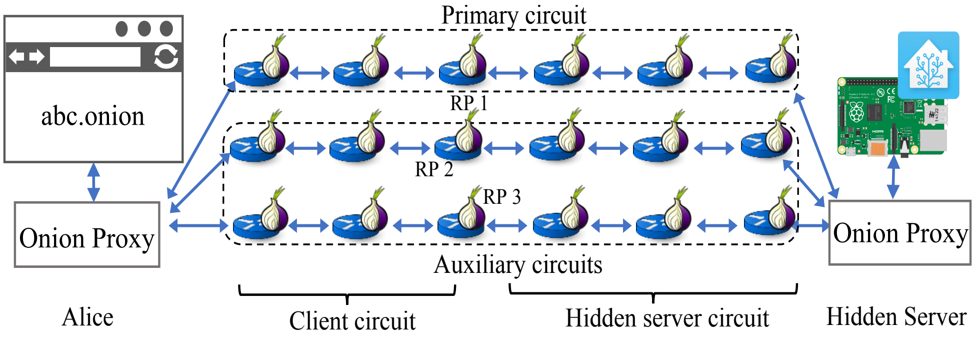


Figure 3.6: An example of $m$TorHS architecture where $m = 3$.

is a client-specific parameter. While the capacity of each circuit is dynamic over time, our proposed $m$TorHS scheme can adaptively distribute traffic onto circuits in proportion to their dynamic capacities, and thus avoid the communication being blocked by a congested circuit and achieve an acceptable overall performance. $m$TorHS is transparent to the Tor network, that is, no modifications need to be made on the existing Tor relays. Only the Tor codes for users who decide to use $m$TorHS need to be changed on user side (i.e., Tor onion proxy OP for browser) and hidden server side (i.e., Tor OP for Home Assistant), for associating multiple circuits to a client stream, adding sequence number to data cell, reordering out-of-sequence cells and scheduling cells across multiple circuits. Next, we elaborate the process of tunnel construction and data transmission. We call user's onion proxy as OP or Alice interchangeably, and call hidden server's OP as HS in the following sections.

### 3.4.1 Tunnel Construction

The server establishes hidden service and client retrieves service descriptors as same as the current Tor hidden services (i.e., *step 3-6* in Figure 3.5). Our modification starts from *step 4*.

**Tunnel initialization.** Different from the current scheme which randomly selects one router as the rendezvous point (RP), Alice, the client side, chooses $m$ routers and constructs $m$ circuits of 3 hops ending at them. Then, Alice gives $m$ different rendezvous cookies (RC) to each RP, which will be used to link the joining circuits established by the hidden server. To ease the presentation, we denote the first established circuit as the primary circuit and the other $m-1$ ones as the auxiliary circuits. Once $m$ circuits are established, Alice does the same thing as *step 5-6* in Figure 3.5: sends an introduce1 message to an introduction point which will forward it to hidden server with an introduce2 message. The message contains the rendezvous cookie, RP address and the first part of a DH handshake for the primary circuit. Besides them, we also add two new fields to this message, namely, $is\_multipath$ and $tunnel\_width$, which indicate that Alice is requesting to build a multipath tunnel with tunnel width $m$. After receiving the introduce2 message, HS checks if $is\_multipath$ is set. If so, HS generates a unique 32-bit tunnel identifier ($TID$), otherwise, HS follows the current single-path Tor scheme. In response, HS establishes a new circuit to the RP of the primary circuit and sends Alice a rendezvous1 cell containing RC and the second part of DH handshake. If RP successfully verifies RC, it relays the rest of the rendezvous1 cell to Alice with a rendezvous2 cell. Once Alice receives it, she extracts the second part of DH handshake and generates the end-to-end session key. A 6-hop circuit is established between Alice and HS. Now, Alice and HS can communicate with each other through the primary circuit.

Then, Alice sends a multipath_m[1] cell to HS through the primary circuit to retrieve the tunnel ID. With $TID$, Alice adds the remaining auxiliary circuits to the tunnel by sending $m-1$ next_rp_m messages to HS along the primary circuit. The format of each next_rp_m message is similar to the introduce1 message, which contains $TID$ and RP's address used in auxiliary circuits. In response to the next_rp_m message, HS builds a new circuit connecting to the corresponding RP, and knowledges each successful joining with a rendezvous1 to that RP. HS associates all circuits with the same $TID$ to form a tunnel for Alice. After Alice receives all $m$ rendezvous2 cells including 1 cell from the primary circuits and $m-1$ cells from the auxiliary circuits, a multipath tunnel is successfully constructed.

**Tunnel management.** Atop circuit management of Tor, $m$TorHS introduces additional tunnel management to avoid the congested circuit degrading the tunnel performance. $m$TorHS manages the multipath tunnel dynamically according to the congestion status of member circuits over time.

[1] To distinguish from the commands in current Tor, all the newly added commands in $m$TorHS have a suffix $m$.

If OP detects that the transmission on a member circuit becomes very slow, it will construct a new circuit to replace the slow one. We will discuss how to define “slow” in next section. The slowcircuit-closing scheme provides OP the ability of responding real-time network dynamics, which prevents a slow circuit from becoming a bottleneck of the entire tunnel. In particular, OP can add new auxiliary circuits or tear down any existing circuit as follows: a new auxiliary circuit can join the tunnel by sending a next_rp_m command to inform HS of the new RP address. To tear down a circuit, OP informs HS to drop it using a drop_m message. After receiving drop_m cell, HS immediately stops sending on this circuit and responds to OP using a dropped_m message with the number of cells that have already been sent on this circuits (denoted as $n_s$). Once OP receives $n_s$ cells on this circuit, it tears down the circuit.

### 3.4.2 Data Transmission

When Alice’s stream arrives at OP via SOCKS, OP spawns the client stream (denoted as the *parent* stream) to $m$ subflows and appends them to the tunnel by associating each subflow with a member circuit. Each subflow has its own stream window and inherits a common stream ID from the parent stream. Next, OP sends a relay_begin cell through a randomly selected member circuit to start the access.

**Scheduling and data cell allocation.** Conceptually, data cells can be forwarded through any member circuit in the tunnel. However, if the number of allocated cells on a particular circuit exceeds its capacity, it will become congested. Since the overall performance of a tunnel is bounded by the slowest circuit, two endpoints of a tunnel need to cooperate to schedule cells across multiple circuits based on the capacities of individual circuits. A straightforward approach for cell allocation is to probe the capacity of each circuit after it is initiated and schedule traffic in proportion to the probed capacity. However, the method is unrealistic and less accurate. First, it will introduce a large amount of extra traffic to the Tor network. Moreover, the capacity of each circuit may change dramatically after the probing. Therefore, the static scheduling scheme cannot adapt to network dynamics which makes the allocation less useful. In [28], Alsabah et al. presented an opportunistic probing approach to estimate the round-trip-time (RTT) of a circuit based on Tor's circuit-level congestion control scheme, where RTT is measured as the difference between the time of sending out every $100^{th}$ cell and the time of receiving a circuit SENDME flag. The number of cells allocated to a circuit is reversely proportional to the measured RTT. The RTT-based approach is reactive to network dynamics, but it is still not very accurate because the congestion feedbacks are received infrequently (every 100 cells) [30]. Besides, cells may spend several hundred milliseconds in the circuit queue before being transmitted over a TCP connection [139], after which the circuit's capacity may change non-negligibly. Based on this consideration, we argue that RTT-based approach may not be a good choice in cross-layer scheduling, which is also recognized in multipath TCP design [35]. In *m*TorHS, we adopt a "pulling" scheduling – instead of pushing data cells to circuits by a scheduler, we let each subflow actively pull data from a shared send buffer whenever its stream window becomes nonempty. Initially, each subflow has a stream window of 500 cells. As described in Section 3.2.2.3-**Flow Control**, the stream window decreases by one when sending a cell out and increases by 50 when receiving a stream-level SENDME notification. Consequently, a subflow stops sending cells when its stream window size drops to zero and resumes when it receives a SENDME. It is obvious that when the circuit to which a subflow is appended becomes congested, cells will be moving much slowly towards the receiver resulting in delayed stream-level SENDMEs and long waiting at the sender end. Whereas, subflows on fast circuits will send out data cells fast and steadily. In this way, the "pulling" scheduling is subflow self-adaptive without accurate explicit circuit RTT measurements. When multiple subflows have a nonzero stream window, we adopt a FIFO (first-in-first-out) queue to schedule them.

**Slow circuits detection.** Another challenging issue in *m*TorHS design is the detection of slow circuits. To avoid a congested circuit becoming the bottleneck of the entire tunnel, OP will construct new circuits to replace the slow ones. In this work, we adopt a distance-based outlier detection approach to determine whether a circuit is congested based on a sliding window of 50 cells. In particular, we measure the time of receiving every 50 cells and find the lower and upper quartiles ($Q_1$ and $Q_3$) of ten most recent records to calculate the interquartile range (IQR) where $IQR = Q_3 - Q_1$.

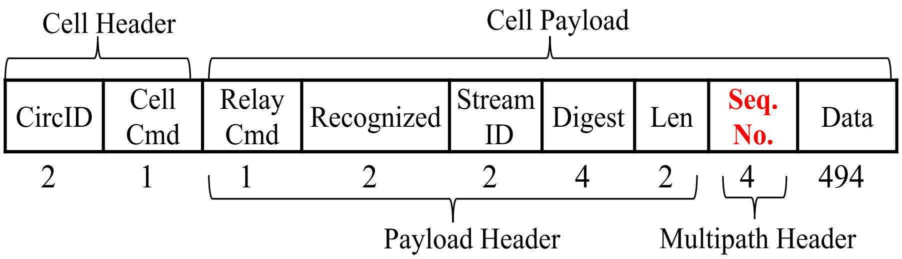


Figure 3.7: New cell format: a new filed, *sequence number* marked in red, is added as the multipath header, representing the sequence number of sent-out cells.

If a new measurement falls out of the range of $(0, Q_3 + 1.5IQD]$, it is considered as an outlier indicating the circuit is experiencing a congestion. To increase detection reliability, OP considers a circuit as congested if at least three consecutive outliers occur. Once detected, OP and HS will collaborate to tear down the congested circuit and replace it with a new one. This can be done through the tunnel management discussed above.

**Data re-ordering.** Combining the self-adaptive "pulling" scheduling and active congestion detection schemes, *m*TorHS is able to adapt to network dynamics, which potentially prevents slow circuits from degrading the overall performance of multipath tunnel. However, due to dynamic scheduling, data cells may arrive at the receiver out of order. To solve this issue, we have to modify the format of Tor data cell to incorporate a 32-bit sequence number in the multipath data packets. As shown in Figure 3.7, the first four bytes of data payload is reserved for this purpose. Moreover, we add a new relay_subdata_m command to indicate a data cell is multipath data. When OP receives a multipath data cell from a subflow, it first checks if the sequence number is what it expects. An expected cell is immediately forwarded to the application stream, while an out-of-order cell is stored in a buffer and ordered according to its sequence numbers.

### 3.4.3 Discussion

By applying congestion detection, users of multipath hidden services can route their traffic through other circuits having light load and thus improve the overall performance. Such a congestion avoidance also benefits single-path users, since it yields the bandwidth on the congested path so that it can be used to relay others' traffic. In terms of security, this solution, namely IoT gateway over multipath Tor hidden services, transmits all traffic including incoming and outgoing traffic on gateway through Tor network, so the IP address of gateway is still hidden from the public Internet and thus the adversary cannot scan and attack the gateway.

Multipath hidden services can improve the performance of a single user, but it will not increase the overall bandwidth of Tor network. If millions of users access their gateways via Tor, especially for bulk traffic like watching camera videos, the huge demand on bandwidth may exceed the capacity of Tor. As a result, even if they all use multipath schemes, the performance eventually will become very poor. Therefore, IoT gateway over multipath Tor hidden services cannot be a viable solution for all users. Instead, it suits those users who have very strong security requirement but only need an acceptable performance. For the majority users who want to achieve a better balance between security and performance, we propose an alternative solution, namely IoT gateway with separate command and data channels, which will be presented in next section.

## 3.5 The Improved Solution: IoT Gateway with Separate Command and Data Channels

Tor provides very strong security protection but poor performance, while the public Internet has the opposite properties. We propose a novel solution, namely IoT gateway with separate command and data channels, which combines the advantages of Tor and the public Internet. More specifically, the gateway only accepts incoming traffic from the Tor channel, while responds to the remote access with outgoing traffic through the Internet channel. This scheme is based on the fact that the IoT gateway can defend against the vulnerability scanning by refusing responding to any request coming from the Internet, such as ICMP ping, telnet, or HTTP request. The adversary obtains nothing by scanning the gateway with IP address. Besides, since the onion address of hidden service is only known by the user himself, the adversary cannot send any scanning traffic to the gateway through Tor. Therefore, the security is still well protected. On the other hand, the

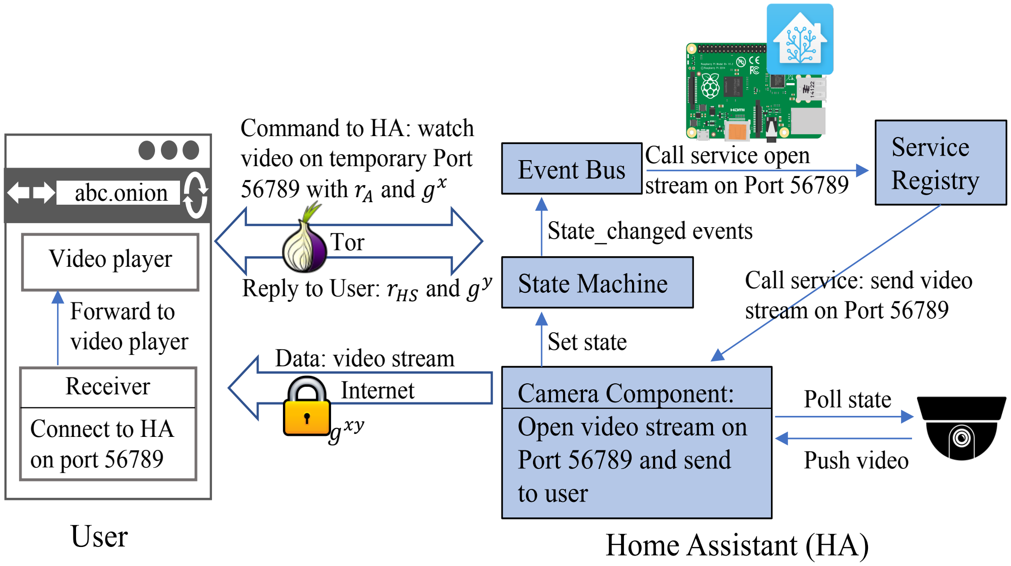

abc.onion
Video player
Forward to video player
Receiver
Connect to HA on port 56789
User
Command to HA: watch video on temporary Port 56789 with $r_A$ and $g^x$
Tor
Reply to User: $r_{HS}$ and $g^y$
Data: video stream
Internet
$g^{xy}$
Event Bus
Call service open stream on Port 56789
Service Registry
State_changed events
State Machine
Call service: send video stream on Port 56789
Set state
Camera Component: Open video stream on Port 56789 and send to user
Poll state
Push video
Home Assistant (HA)

Figure 3.8: An example of IoT gateway with separate command and data channels: user requests to watch camera video on temporary port 56789 through the Tor channel, while Home Assistant temporarily opens port 56789 and sends video to user from the requested port through the Internet channel.

incoming command is usually transient and very small, so transmitting the command through Tor will not introduce much extra overhead to Tor. In response to the command coming from the Tor channel, the gateway sends out bulk traffic such as camera videos to the user through the Internet channel. Since the video stream is transfered through the public Internet, we need to make sure that user and Home Assistant are mutually authenticated and the traffic is encrypted. With separate incoming command channel and outgoing data channel, we can leverage both the security of Tor and the good performance of the public Internet. Next, we will use an example that user requests video stream to demonstrate our design, which is shown in Figure 3.8. The interactive protocol between user and Home Assistant is as follows:

**Step 1: User initiates a connection through Tor**. To access Home Assistant, user types its onion address (e.g., abc.onion) in the mobile application. After user connects to the control panel through hidden service, she specifies a random high TCP port, a 256-bit random token $r_A$, a 1024-bit random number $x$ and its corresponding Diffie-Hellman number $g^x$ in the camera dialog. The port number will be used by Home Assistant to send video to user, the random token $r_A$ will be used for mutual authentication later, and the Diffie-Hellman data $g^x$ will be used to generate session key. Then, user sends this command to Home Assistant through Tor.

**Step 2: Home Assistant gets the requested service ready**. In the original Camera Component of Home Assistant, the video received from the camera will be shown on the web interface, and thus, in response to the request from the Tor channel, the video is also directly sent back to the user through Tor. In our scheme, we rewrite the Camera Component so that it can redirect the video stream to a temporary port specified by the user and forwards the stream from that port to user through the Internet channel. To do so, we just add a new functionality to the Camera Component and register this service to Service Registry. Since this new service is implemented on the component level, it can work with different types of cameras.

When Home Assistant receives user's command from the Tor channel, it generates a 256-bit random token $r_{HS}$, a 1024-bit random number $y$ and its corresponding Diffie-Hellman number $g^y$. HA uses the received $g^x$ and $y$ to generate the session key $g^{xy}$ and responds to user with the random token $r_{HS}$ through the Tor channel. Meanwhile, HA opens port 56789 and waits for the connection for video streaming. Note that the socket that listens on port 56789 is set to accept only one connection, and close after the connection is finished. Now, Home Assistant is ready to receive request from user through data channel.

**Step 3: User requests data through the Internet**. After user receives the random token $r_{HS}$ and the second half Diffie-Hellman handshake data $g^y$ through Tor, she will generate the session key $g^{xy}$, and sends a request containing $r_{HS}$ to Home Assistant through data channel, which is encrypted with the session key $g^{xy}$.

**Step 4: Home Assistant authenticates the request and sends encrypted data**. Once user's request for video stream arrives at port 56789 through the Internet channel, Home Assistant first decrypts it with the session key and verifies whether the token in this request matches the previous one that it sends to user. If the verification succeeds, HA sends $r_A$ and the subsequent steaming data to user, which will be encrypted with the session key. Otherwise, HA discards the request without any reply and closes the port 56789. Note that this request containing the authentication data is the only packet that HA will accept. After that, HA will not receive any incoming packet from port 56789 except ACK.

**Step 5: User authenticates the reply and receives data**. After user receives the reply and can correctly decrypt it to get $r_A$, she will accept the subsequent data. Otherwise, the connection is closed. After user stops receiving data from HA, this connection will tear down and port 56789 on HA side is also closed.

## 3.6 Performance Evaluation

To explore the performance improvement introduced by our two schemes, we perform experiments on the live Tor network and compare the average time of video stream transmission between the original Home Assistant without Tor hidden services (denoted by *HA-No-Tor*), HA with single-path hidden services (*HA-sTorHS*), HA with multipath hidden services (*HA-mTorHS*) and HA with separate command and data channel (*HA-Separation*).

**Setup.** We turn a Raspberry Pi 2 [18] into a dedicated Home Assistant hub, which connects to a VStarCam IP camera [22] where the video stream is feed into Home Assistant via FFmpeg. The Raspberry Pi 2 is equipped with 700MHz ARM A6 microprocessor and 512 MB RAM. A client accesses to Home Assistant through a laptop with 2.5GHz Intel i5 CPU, 8GB RAM and OSX. To compare the performance, we measure the time that client receives 10MB streaming data from Home Assistant. For *HA-Separation*, our modification to Home Assistant is made using Python 3 on the backend and Polymer on the frontend. For *HA-mTorHS*, we change the source code of Tor-v0.2.9.10. Note that all changes are made to the client's proxy and hidden server's proxy, so no modification is needed on the Tor network. We compare different multipath settings where the tunnel width $m$ is set to 2 and 4, respectively. To eliminate the difference of different circuits' capacities used by different schemes, we ask *HA-4TorHS* scheme to use the default path selection algorithm to choose relays for 4 circuits and record the used relays. Then, *HA-2TorHS* scheme randomly chooses 2 out of 4 circuits, while *HA-sTorHS* randomly chooses 1 from 4 circuits.

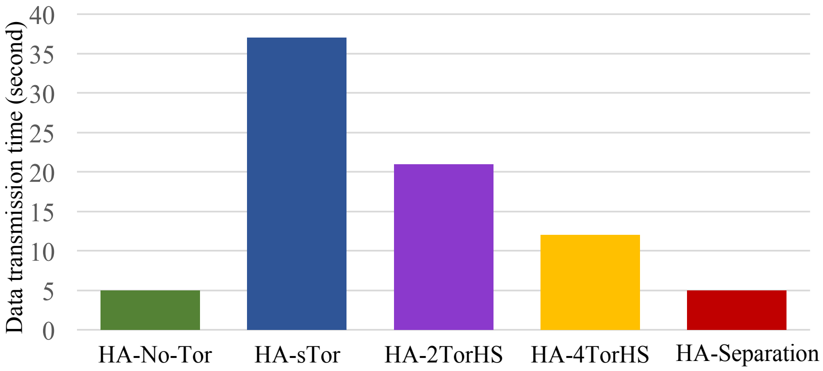


Figure 3.9: Comparison of transmitting 10MB video streaming data between user and hidden server (i.e., Home Assistant).

**Results.** Figure 3.9 compares the performance of different schemes. The Y-axis is the average time of transferring 10MB video data after we run the experiment 60 times over different time of a day. From the figure, we can see that the current proposal (blue bar in Fig. 3.9) that directly integrates Tor hidden services into Home Assistant achieves the worst performance, which is almost 7 times slower than the direct access to HA through the public Internet (green bar). In contrast, our proposed multipath Tor hidden services improve the performance greatly: *HA-2TorHS* having 2 circuits is 1.7 times faster than *HA-sTorHS* , while *HA-4TorHS* is 3 times faster than it. As we discussed in Section 3.5, to further improve the performance and alleviate the overload on Tor network, most users should adopt *HA-Separation* (red bar), which can achieve the same performance as directly accessing to Home Assistant without using Tor.

*m*TorHS improves the performance, because it can overcome the problem that a single path gets congested. To verify this, we test a case where the 6-hop path between client and hidden server is established using very fast relays, but the path is congested, meaning that the performance is much worse than the average time. For *HA-sTorHS*, 3 relays used by user are 2391A, 92CFD and 3E13E, while 3 relay used by HS are A7047, 96DAF and 8C23B. They are respectively Top $1\%, 5\%, 14\%, 30\%, 11\%$ and $20\%$ fastest relays in all Tor routers at measurement time (July 12, 2017). This path consisting of very fast relays should have provided very good performance, but its real performance is 55 seconds, which is much slower than the average of *HA-sTorHS*. The poor performance on this path are usually due to congestion on at least one of relays in the path [150]. To see the effect of our multipath scheme, for *HA-2TorHS*, we do not change the first path and add another less congested path. The performance is improved to 16 seconds, since most traffic can be routed through the second path, which also alleviates the congestion on the first path. For *HA-4TorHS*, we add another 2 less congested paths, and the performance is further improved to 10 seconds.

## 3.7 Security Analysis

In this section, we analyze the security of two proposed schemes in terms of authentication, encryption and anti-scanning.

**Authentication**. Although Home Assistant itself supports optional password-based authentication, it does not enforce users to apply it. Besides, such an authentication approach has numerous deficiencies. The most well know problem is weak password. Since hackers are not forbidden on unlimited attempts at guessing the password through the public Internet, the authentication may be compromised. Therefore, the authentication provided by Home Assistant is not reliable. In contrast, besides password required by HA, our proposed two schemes offers two additional layers of authentication with Tor hidden services. First, adversaries cannot access Home Assistant over hidden services without knowing the onion address. Since onion address generated by Tor is an 80-bit number in base32, it is not easy for adversaries to predict the one used by a target. In addition, even if the adversary obtains the onion address by chance or does brute force to attempt all onion addresses, Tor hidden services also require users to have a 132-bit authentication cookie in base64 to access the hidden server. It is very difficult for adversaries to guess the correct combination of onion address and authentication cookie. Therefore, our schemes which integrates Tor hidden services can provide a very strong authentication for IoT gateway. In *HA-Separation* scheme, when user connects to HA to receive data through the public Internet, a mutual authentication is enforced by verifying random numbers exchanged through Tor. Because those two random numbers transferred through the Internet are encrypted by session key, they cannot be forged by adversaries, and thus the mutation authentication for the data channel is reliable. In addition, since the socket on HA is set to accept only one connection, adversaries cannot do unlimited attempts. Note that the secret onion address and the authentication cookie are stored in the Tor configuration file on client's device , such as his laptop or smart phone. Our schemes cannot defend against the attack that an adversary can hack into client's device to steal onion address and authentication cookie. This is a very complex problem about how to protect the secret key on a laptop or smart phone from cyber attacks, which is out of the scope of this work. Besides, since we assume that smart home owners will not share the secrets including onion address and authentication cookie to many people except family members, it will not be a difficult task to manage them and keep them secret.

**Encryption**. Home Assistant does not use HTTPS by default, so it is very insecure to access it remotely. To solve it, users are suggested to set up encryption using Let's Encrypt [14]. However, this requires a tedious process to finish the setup, especially for users who know little about network and security, which usually discourages users to use it in reality. In our schemes, since Tor hidden services have built-in onion encryption and end-to-end encryption, once it runs for IoT gateway with a very simple configuration, all traffic that goes through Tor is well protected without user's involvement any more. In our *HA-mTorHS* scheme, since all communication is over Tor, the data confidentiality completely relies on Tor's strong cryptography technologies. In our *HA-Separation* scheme, we use Tor channel for command and Internet channel for data. All commands are protected by Tor as discussed before, while data is also encrypted by an AES session key, which is negotiated through the command channel between user-end application and Home Assistant over hidden service.

**Anti-scanning**. As we discussed in Chapter 2, anti-scanning approach is a very effective solution to cyberattacks on smart home. The current Home Assistant running on default port 8123 will respond to adversary's scanning on this port, so the adversary can find vulnerabilities and exploit them. In our *HA-mTorHS* scheme, all accesses to Home Assistant must go through hidden service. Without knowing the onion address and authentication cookie, adversary does not know the existence of Home Assistant, and thus cannot probe and access it. In our *HA-Separation* scheme, authentication and key negotiation are conducted through the command channel over Tor, so those vulnerability scanning techniques discussed in Chapter 2 will not work any more. For data channel, it is also resistant to scanning when the port is temporarily open to the public Internet for data transmission due to two reasons: (1) Since the port on Home Assistant is a random port and only temporarily open during data transmission, the adversary even may not have enough window of time to detect a open port via massive scanning, let alone a successful attack; (2) Even if we assume that the adversary controls a large number of machines and is able to scan all ports of an IP address, the adversary only knows that a high port, e.g., port 56789, is open, but nothing else. That is because after the TCP handshake, Home Assistant accepts only one authentication data packet that contains the nonce encrypted by session key. Any other traffic will be dropped, so the subsequent scanning for detailed information such as version, OS and model, will receive no response.

## 3.8 Related Work

All related work to this paper can be roughly classified into two categories:

**Countermeasures enhancing IoT security**. Extensive research has been done to enhance the security of individual devices. For authentication, Liao et al. [105] propose a secure ECC-based RFID authentication scheme to realize the mutual authentication between devices. Wu et al. [155] further improve the security by proposing lightweight private mutual authentication and private service discovery. For encryption, traditional cryptography can be applied to secure IoT. Dinu et al. introduce a benchmark framework to evaluate how well lightweight block ciphers, such as AES, RC5, Simon and Speck, etc., are suited to IoT devices. For communication, a bunch of dedicated protocols such as CoAP [131], RPL [152], and 6LoWPAN [130] and their variants have been proposed, which are not only lightweight for IoT devices but also security-oriented. Another type of approach that achieves both security and lightweightness is cloud-assisted IoT security. Since resource-constrained IoT devices usually cannot afford costly cryptographic techniques and large data storage, many schemes are solving this problem by leveraging cloud which provides powerful computation and storage capacity [83, 157, 163, 86]. [17, 29] both focus on cloud-assisted healthcare IoT, which mainly use the storage resources of the cloud. In [17], they proposed a scheme to add watermark into the collected data of a patient to avoid the privacy leakage on the cloud, while Yang et al. proposed a scheme that allows health service providers such as doctors to access and verify the encrypted medical records stored on the cloud by using a searchable encryption with forward privacy support [29]. In contrast, [28] utilizes the computation resources of the cloud to implement a data publishing scheme adopting attribute-based encryption, while [86] proposes a data access control scheme for constrained IoT devices and cloud computing based on hierarchical attribute-based encryption. The above solutions mainly focus on securing individual IoT devices, while IoT gives us a way to manage and secure a bunch of heterogeneous devices. For example, Intel IoT gateway can support comprehensive device protection with integrated McAfee, including secure boot, application integrity monitor, encrypted storage and more [11].

**Multipath Tor schemes**. Several multipath Tor schemes have been proposed to solve the performance and anonymity problems of Tor [28, 160, 159]. AlSabah et al. [28] first explore how to use multipath routing to improve Tor's performance, and then Yang et al. [160] further analyze the relay utilization and propose that using low-bandwidth relays to construct multiple circuits can both improve the performance and increase the network utilization. Different from [28, 160] focusing on improving performance for general Tor services, Yang et al. [159] proposes a partial multipath routing scheme for Tor hidden services to enhance the resistance to traffic analysis attacks. In their scheme, the tunnel is built between the rendezvous point and the hidden server. They improve the anonymity based on the insight that traffic pattern will be somewhat distorted by flow splitting and flow merging operations in multipath routing and by the multiple routes with different network dynamics. In contrast, with the goal of improving the performance, our proposed multipath Tor hidden services for IoT gateway adopts an end-to-end multipath structure, which leverage the end-to-end traffic management to overcome the network dynamics. Another notable difference between the above schemes and ours is that our multipath scheme is transparent to Tor network, namely, no modifications are required on existing Tor routers except user proxy who is using multipath Tor, and thus, our scheme can be used immediately, while all the above three schemes require new Tor routers to support their designs.

## 3.9 Summary

In this chapter, we discuss the threats on smart home and analyze the challenges of realizing countermeasures against those threats. To tackle those problems, we propose two novel Tor hidden services based solutions to secure smart home: IoT gateway over multipath Tor hidden services and IoT gateway with separate command and data channel. In the first solution, we use multipath routing scheme to improve the performance of Tor hidden services. With all data routed in the Tor network, this solution suits users who desire strong security but acceptable performance. To improve the performance, our second solution transfers command through Tor and transfers data through the Internet so that we can leverage both the strong security of Tor and the good performance of the Internet. Our experimental results and security analysis show that the proposed two schemes are promising solutions for securing smart home in practice.

# Chapter 4

# Enhancing Traffic Analysis Resistance for Tor Hidden Services with Multipath Routing

## 4.1 Introduction

In the previous chapter, the goal of using multipath routing scheme is to improve the performance of Tor hidden services for users who require strong security and acceptable performance. However, another well known problem of Tor hidden services is the security problem, namely, Tor hidden service is vulnerable to traffic analysis. As a result, once adversaries can find the existence of hidden service for smart home gateway, the gateway will become the target of adversaries and thus the protection on home gateway may fail. Therefore, the questions that we ask in this chapter are: What if smart home user wants stronger security? Can we still improve the anonymity of Tor hidden services and further improve the security of smart home gateway? In this chapter, we will answer those questions still using multipath routing technique but in a different way.

According to recent studies Tor hidden services are still under the risk of de-anonymization due to specialized traffic analysis attacks [121, 38]. It is argued that the current Tor design is vulnerable to traffic analysis attacks if the adversary can monitor a user's traffic entering and leaving the anonymous network at both sender and receiver ends. Since the malicious client always resides at one end of the anonymous path, she can successfully perform the traffic analysis attack if she is able to observe the traffic at the hidden server end. Øverlier et al. proposed the first documented attack against Tor hidden services by exploiting traffic analysis techniques. They experimentally verified that a hidden server can be located within a short period of time if the adversary is able to control one Tor (or preferably two) router(s) [121]. Biryukov et al. also confirmed the practicality of traffic analysis attacks by conducting an opportunistic de-anonymization attack to Tor hidden services [38]. The effectiveness of such attacks is mainly caused by the low latency in anonymized paths, which unwillingly preserves the inter-cell timing correlation between the original flow and the anonymized flow. The adversary can exploit traffic analysis techniques to correlate common patterns between the original flow and the anonymized flow to infer identities and relations of the communicating parties. Therefore, the key to mitigating the threats of traffic analysis attacks is to reduce the timing correlation between cells. Dummy traffic is considered as an effective countermeasure to obscure the timing features of the original flow [132]. However, due to the high cost introduced by dummy traffic, it is not a practical solution for the already heavily loaded Tor network.

In this chapter, we propose a multipath routing scheme for Tor hidden services (*m*TorHS-S) to defend against traffic analysis attacks (To distinguish the multipath Tor hidden services aiming to improve performance in the previous chapter, we add an "S" to this scheme, representing that the goal of this scheme is to improve security). Our scheme routes data cells between the rendezvous point and the hidden server through multiple circuits, which exploits flow splitting and flow merging functionalities of multipath routing to remove identifiable patterns of the original flow. Through experiments on the Shadow simulator [88], we show that *m*TorHS-S is resistant to traffic analysis, even when robust watermarking-based techniques are employed. In addition, by integrating multi-flow detection scheme [96] into *m*TorHS-S, our scheme is able to combine multiple watermarked flows to detect the presence of watermarks, if they have not been completed destroyed by multipath routing.

| Abbreviation | Term | Abbreviation | Term |
|---|---|---|---|
| HS | Hidden server | SRP | Server's rendezvous point |
| Alice | Malicious client | RC | Rendezvous cookie |
| RP | Rendezvous point | DH | Diffie-Hellman |
| OR | Onion router | IP | Introduction point |

Table 4.1: Definitions of abbreviations used in this chapter.

Because a large number of abbreviations are used in this chapter, we summarize the notions in Table 4.1. The remainder of this chapter is organized as follows. After introducing two representative traffic analysis attacks against Tor hidden services in Section 4.2, we present the threat model in Section 4.3. Then, we elaborate the detailed design of our multipath Tor hidden services in Section 4.4. In Section 4.5, we experimentally evaluate the effectiveness of *m*TorHS-S against a very robust watermarking-based attack. Finally, we summarize this chapter in Section 4.6.

## 4.2 Traffic Analysis Attack against Hidden Services

It is recognized that Tor hidden service is vulnerable to traffic analysis attacks. In general, traffic analysis attacks can be classified into two categories: *passive* traffic analysis and *active* traffic analysis. Passive traffic analysis correlates the sender's outgoing traffic with the receiver's incoming traffic by comparing the traffic features, such as packet timings and counts. To launch a successful passive traffic analysis, the adversary needs to monitor the traffic for a long time to obtain a reliable traffic pattern. The biggest advantage of passive traffic analysis is its stealth, but it is time-consuming and less accurate compared with the active attacks. To improve the accuracy and reduce the cost, many active traffic analysis techniques have been proposed to generate traffic with a special pattern at one end of the communication path and identify it at the other end.

The security of Tor hidden services was first challenged by Øverlier et al. [121]. They experimentally attacked an early version of hidden services in which the entry guard protection mechanism has not yet been implemented in Tor. In response to a client request, HS will randomly select three routers to build a circuit to RP. Assume a malicious client controls a set of routers in the Tor network. By establishing a large number of connections to HS, she can eventually force HS to choose an entry router (i.e., the fist hop of a circuit) that she controls. Then by requesting files of different sizes at different time from HS, the attacker can generate a special traffic signature and exploit simple traffic analysis techniques (e.g., packet counting combined with timing information) at the malicious entry node and the RP to correlate flows with the same traffic pattern.

Another attack is proposed by Biryukov et al. [38]. They generated traffic with a special pattern and applied packet counting traffic analysis to identify flows with the injected pattern. For example, a malicious RP can send 50 *padding* cells and a *destroy* cell to HS after receiving the rendezvous cell in Step (7) (as shown in Figure 3.5). If the corresponding malicious entry guard observes 53 cells (including the destroy cell, 50 padding cells and 2 additional *extended* cells in circuit construction) going towards HS and 3 cells (including the rendezvous cell and 2 *extend* cells) leaving HS, the adversary can decide that this malicious guard node is chosen as the entry node by HS.

However, these two attacks also suffer drawbacks. Since Øverlier's attack was conducted in the early stage of Tor with much fewer routers, clients and hidden servers, at that time their generated traffic pattern was unique enough and hence can be preserved after going through the Tor network. Nevertheless, the current Tor with much more traffic will make this simple packet counting based analysis less effective. For Biryukov's attack, because special cells (i.e., *padding* cells) are used to generate a unique traffic signature, it may not be invisible to hidden server. Therefore, more advanced active watermarking-based traffic analysis techniques [151, 109, 85, 84] are proposed, which can make the traffic analysis attacks targeting Tor hidden services more efficient and stealthy. They embed a specific traffic pattern to the victim's flow on the sender side by manipulating the timings of selected cells. The adversary breaks the anonymity guarantee if the watermark is uniquely identified on the receiver side. Compared to passive traffic analysis and other active traffic analysis techniques, watermarking is more robust to flow transformations such as dummy traffic, flow mixing, traffic padding and network jitter, so it is considered a more efficient and severe threat

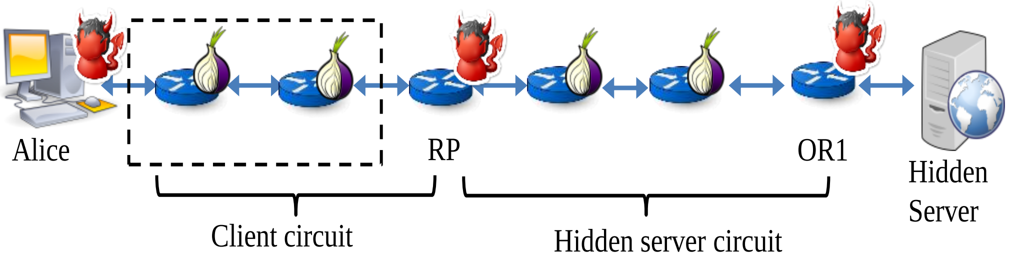


Figure 4.1: Threat model in this chapter where RP and OR1 are controlled by the adversary

to Tor hidden services.

## 4.3 Overview of the Problem and the Threats

The attacks described in Section 4.2 show that the adversary can successfully correlate two communicating parties if she is able to observe the traffic at two ends of a Tor circuit. As shown in Figure 4.1, the anonymous path between a client and the hidden service server consists of 6 hops. Since a malicious client is always at one end of the path, she only needs to trick HS to choose a compromised router controlled by herself, i.e., OR1. Her success rate relies on the proportion of compromised routers in the Tor network. Therefore, this attack particularly threatens the hidden services. Moreover, the adversary can further select a node that she controls to be the rendezvous point and build a one-hop circuit to this RP. In this way, she can shorten the path to four and thus reduce the latency between herself and HS to help correlate the traffic pattern. To mitigate this threat, efforts can be made from two perspectives: (1) preventing an adversary from controlling both ends of a circuit to impede the occurrence of traffic analysis (2) reducing the success rate of traffic analysis even when both ends of a circuit are compromised.

The concept of "entry guards" [121] is introduced into the current Tor design to solve this problem following the first direction. Entry guards are a set of routers that are considered reliable by a Tor node to be the first relay of an anonymous path. By default, each user constructs its guard set of three routers, which will expire in 30 to 60 days. After that, the entry guards will be reselected. With entry guards, whenever the hidden server builds a circuit to the rendezvous point in response to a client's request, it will pick an entry guard from the set for its first hop instead of choosing a random router in the network. Since the entry guards are evaluated by several measures and considered reliable, they are less likely to be controlled by the adversary. As a result, the chance that an adversary controls both ends of a circuit is significantly reduced. However, it is unreliable that the security of hidden servers merely relies on the goodness of the entry guard set. Given enough time, a user will eventually select a malicious entry node into his guard set. Johnson et al. showed that for an adversary with moderate bandwidth capacity, it only takes 50 to 60 days to include a malicious router to a user's guard set [92]. As noted by Elahi et al., the design of entry guard is still an unclear research problem [62] and subtle parameter selection is required to achieve expected protection. Therefore, it is critical to develop protection mechanisms which are parallel to the entry-guard-based solution to enhance the resistance of Tor hidden services to traffic analysis attacks in case the guard set is compromised.

In this chapter, we make efforts following the second direction to reduce the success rate of traffic analysis when the attacker has successfully controlled both ends of a Tor circuit. Our work can be applied in concert with entry guard protection mechanism. Figure 4.1 illustrates our threat model. We assume an adversary Alice pretends to be a client of the hidden server and tricks the hidden server to select a node controlled by her (OR1) to be the entry node in the return anonymous path. Alice then selects another controlled router as the rendezvous point. We assume Alice can exploit any traffic analysis technique to passively observe or actively manipulate the traffic passing through OR1 and RP. The primary goal of the traffic analysis is to identify flows with the same traffic pattern at OR1 and RP to confirm that both malicious nodes are recruited in HS's circuit, from which she can learn the location of the hidden server.

## 4.4 Multipath Tor Hidden Services for Security (*m*TorHS-S)

To prevent the adversary's RP and entry node from correctly identifying the traffic pattern, we present a multipath routing scheme for Tor hidden services. This scheme is based on the key

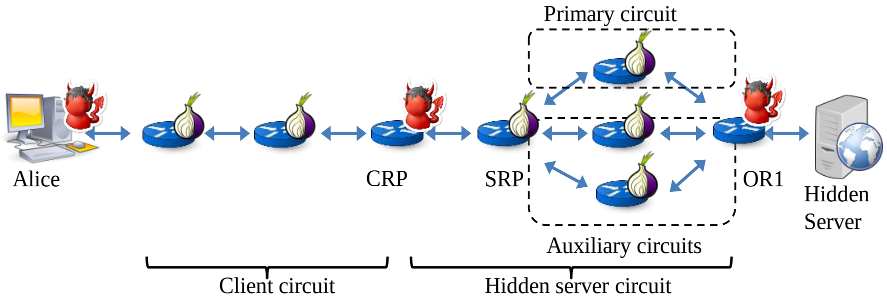


Figure 4.2: *m*TorHS-S architecture

insight that the traffic pattern observed or intentionally generated at the malicious entry guard (e.g., OR1) will be somewhat distorted by flow splitting and flow merging operations in multipath routing and by the multiple routes with different network dynamics. The architecture of *m*TorHS-S is illustrated in Figure 4.2. Different from the selection of rendezvous point in the current Tor implementation, the hidden server also selects its own "rendezvous point". To distinguish two rendezvous points, the one selected by the client is denoted as "CRP" and the one selected by hidden server is denoted as "SRP". In respond to a client's request, HS builds an anonymous tunnel consisting of *m* circuits, where *m* is a server specific parameter. HS then splits the original flow onto *m* subflows and attaches each subflow to a circuit in the tunnel. All *m* circuits will go through the same entry guard OR1 and merge at SRP, which further relays the merged flow towards the client. Next we will present the detailed process of connection initiation and data transmission.

### 4.4.1 Connection Initialization at Client Side

For the client, the connection initialization remains the same as the current Tor hidden services (i.e., *step 3-6* in Section 2.2). More specifically, Alice first selects a client rendezvous point (CRP) and constructs a rendezvous circuit to it by sending an establish_rendezvous request. Then, she builds an introduce circuit to one of HS's introduction points and sends an introduce request, which requests for the hidden service at HS and informs HS with CRP's address (i.e., fingerprint).

### 4.4.2 Multipath Tunnel Construction on Hidden Server Side

After receiving the introduce request, HS decrypts it with its private key and extracts the fingerprint of CRP, rendezvous cookie (RC) and the DH handshake message. Then, HS selects its own rendezvous point (SRP) and constructs a multipath tunnel to SRP, following the same approach described in [161].

**SRP selection.** The selection of SRP is very critical. From Figure 4.2, we see that subflow merging occurs at SRP. Hence, even when multipath routing is adopted between SRP and HS, if the adversary controls SRP and OR1, she can observe traffic patterns before merging from both ends of each subflow and thus perform traffic analysis successfully. The adversary may follow the same strategy as described in [121] to trick HS into selecting a controlled node as SRP by continuously sending a large number of requests. If HS selects a new SRP for each received access request, it may eventually select one of the compromised router. Inspired by the entry guard idea, we propose "rendezvous guard" for SRP selection, which is a set of reliable routers selected by the hidden server. A hidden server initially selects three routers to compose its rendezvous guard set, each of which stays in the set for a random period between 30 and 60 days. Whenever HS builds a rendezvous circuit in response to the client request, it sticks to the same rendezvous guard set and randomly picks one router from it.

**Tunnel initialization.** As discussed previously, *m*TorHS-S constructs a tunnel with multiple circuits to SRP instead of one anonymous circuit to CRP. In the original Tor hidden service design, the hidden server responds to an introduce request by establishing a four-hop anonymous circuit ending at CRP selected by the client. To ease the presentation, we denote this anonymous circuit as the primary circuit and the other $m-1$ anonymous circuits in the tunnel as the auxiliary circuits. As shown in Figure 4.2, all circuits merge at the SRP. Therefore, it is the third router for every three-hop anonymous circuit in the tunnel. Besides that, HS follows the default Tor path selection algorithm to select all other routers to form the tunnel. When the circuit is established, HS sends a multipath_m cell along the primary circuit to SRP to request a multipath connection. In response, SRP generates a unique 32-bit tunnel identifier (*TID*) and incorporates it into the replied multipath_ack_m cell to indicate a successful multipath tunnel construction. With *TID*, HS adds each auxiliary circuit to the tunnel by sending a join_m request to SRP along the circuit. SRP acknowledges each successful joining with a joined_m message. Finally, when HS receives $m-1$ acknowledgments, a multipath tunnel is successfully constructed. Note that all the cells are layered encrypted so only HS and SRP at two ends see *TID* and the newly added tunnel construction command. This prevents the entry and middle nodes of HS's circuit from linking *TID* with HS. In fact, they even do not know if they are involved in any tunnel construction.

Once the tunnel is established, HS follows the same process of the current hidden service protocol to extend path construction to CRP and the client. In particular, HS sends a rendezvous1 cell containing DH handshake message and rendezvous cookie (RC) to CRP along the primary circuit, which verifies RC and joins the client's circuit with the server's primary circuit. Then, CRP sends a rendezvous2 cell containing the DH handshake message to the client to finish the construction. Note that from CRP's view, it sees only the primary circuit connecting itself to SRP, and hence it has no idea about how many circuits are involved in the multipath tunnel between SRP and HS.

**Tunnel management.** The hidden server can add new auxiliary circuits or tear down any existing circuit in the tunnel after it is established. In particular, a new auxiliary circuit can join the tunnel by sending a join_m command with the corresponding *TID*. To tear down a circuit, HS immediately stops sending on this circuit and informs SRP to drop it using a drop_m message. Note that the number of cells that have already been sent on this circuit (denoted as $n_s$) should also be passed to SRP to avoid packet loss. After receiving drop_m cell, SRP extracts $n_s$ and replies with a dropped_m cell after it receives the remaining $n_s$ cells. Finally, HS tears down the circuit when it receives the dropped_m message. Since each tunnel is constructed in response to a client's request, it will be closed after the request is completed. However, this will not result in the closure of all circuits in the tunnel, since the circuits may be reused for other purposes until it gets "dirty" - after its lifetime exceeding 10 minutes and no streams on it, similar as in Tor circuit management.

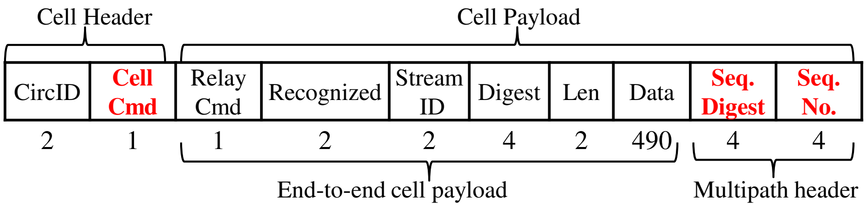


Figure 4.3: *m*TorHS-S cell format

### 4.4.3 Data Transmission between Client and HS

Once the connection is set up, the client and HS can communicate through the anonymous path consisting of the server's and the client's anonymous circuits joined at CRP. Data cells between SRP and HS can be routed through any circuit in the tunnel. To indicate a cell is a multipath cell used in *m*TorHS-S, we add a new cell command (i.e., MULTIPATH_CELL). HS is responsible for assigning data cells to circuits. Obviously, if HS schedules consecutive cells onto a same subflow, it is highly likely that the traffic pattern inserted by the malicious guard on this subflow will be preserved in the merged flow and detected by the malicious CRP. To reduce the likelihood of inter-cell correlations, HS randomly assigns data cells to subflows with different capacities. As a result, a data cell from a fast circuit needs to wait at SRP for earlier cells arriving from slow subflows to be merged in an orderly manner. In this way, we utilize the network properties of different circuits to distort or destroy potential traffic patterns inserted by the malicious guard. This greatly reduces the likelihood of inter-cell correlation (we will explain this in Section 4.4.4).

**Data cell format.** Since the capacity of each circuit in the tunnel varies, different delays will be introduced to these subflows. Therefore, the data cells in different subflows may arrive at SRP out of order. To solve this issue, we modify the format of Tor data cell to incorporate a 4-byte *sequence digest* and a 4-byte *sequence number* for multipath data packets, as shown in Figure 4.3. Originally the 512-byte cell consists of a 3-byte cell header including a circuit identifier and a cell command for cell type, and 509-byte cell payload with a payload header and the payload data. We use 8 bytes of the cell payload as *multipath header* for cell reordering, where 4 bytes are used as sequence digest for integrity check and 4 bytes are used for sequence number. The multipath header is only used by SRP and HS to reorder data cells, and the remaining 501-byte end-to-end cell payload is used to carry the real payload data between client and HS.

**Data cell encryption.** In Tor anonymous routing, data cells are encrypted in layers with the shared session keys of the intermediate relays in the order of their relative positions in the anonymous path. Since the end-to-end cell payload and the multipath header are designed for the client and SRP respectively, HS needs to encrypt the two parts separately. The end-to-end cell payload should be encrypted in *five* layers with the inner-most layer encrypted by the end-to-end session key and the outer-most layer encrypted by the key of OR1, while the multipath header is only encrypted in *three* layers with the keys of SRP, the middle router OR2 and the entry router OR1, respectively. When an intermediate router receives a multipath cell, it applies its secret session key onto both end-to-end cell payload and the multipath header to unwrap one encryption layer. Consequently, at SRP the multipath header will be completely unwrapped and recognized by SRP for further processing, while the end-to-end cell payload is still encrypted and remains secure.

**Data cell reordering.** To merge multiple circuits in a tunnel, SRP orders the received cells from all circuits according to their sequence number and temporarily stores the out-of-order cells in a buffer. When SRP receives a multipath data cell from a subflow, it first decrypts the multipath header and generates a digest for the last four bytes of the multipath header using the symmetric key shared with HS. If it is the same as the received sequence digest, SRP verifies the sequence number is not tampered. If the sequence number of this cell is what SRP expects, it will be immediately forwarded to CRP, otherwise it will be stored and ordered according to the sequence number. The multipath header field is only used for data cell reordering. Hence, after SRP reorders the cells and merges them into one output stream, this field becomes useless. To avoid unnecessary information leak, SRP will replace it with random bits. Similarly, when the client sends data to HS, the client reserves these eight bytes for SRP by padding them with random numbers.

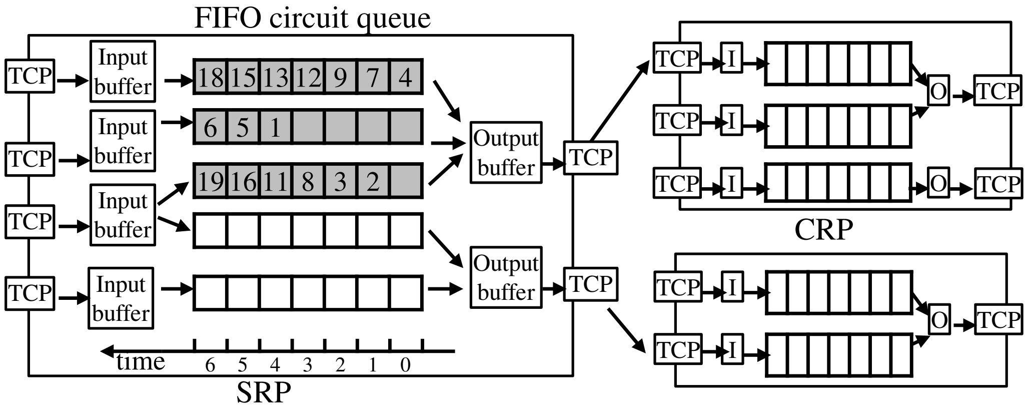


Figure 4.4: Tor router queuing architecture [30]

### 4.4.4 Discussions

In Tor network, two Tor routers are connected over a TCP connection, which is multiplexed by several circuits. Due to the multiplexing of a TCP connection, flow mixing actually occurs at every router. However, we argue that the cell distribution is well preserved after flow mixing so that a maliciously inserted traffic pattern can still be observed by the attacker. First, let us explain the data cell processing at a Tor router. When a cell arrives from a TCP connection, it triggers the *connection read event* of libevent to first put it into the application-layer input buffer and then send to the corresponding circuit queue according to its circuit identifier. As shown in Figure 4.4, five different circuits arrive SRP from four TCP connections. Then, a *connection write event* will select a circuit based on pre-determined scheduling algorithms such as priority-scheduling [139] to pull cells from the circuit queue and send them to the output buffer. As a low-latency system, a router will send out cells in the circuit queue as fast as possible until the output buffer is full. Therefore, it is not surprising that cells from a same subflow will be outputted in a batch with inter-cell features well preserved.

In this chapter, we propose a multipath routing approach that introduces an interdependent subflow mixing to SRP data cell processing. For example, in Figure 4.4 suppose the circuits in gray belong to the same tunnel. Each of them is associated to a subflow, which transfers a portion of data cells. Since the malicious guard has no clue about the flow membership of the subflows passing through it, it has to treat each subflow independently when inserting detectable traffic patterns. Oppositely, SRP will treat subflows of a same flow in a way that considers flow interdependency. In particular, when subflows are merged at SRP, cells from one subflow may be inserted into two cells that are adjacent in another subflow. This interpolation causes difficulty in pattern detection on the merged flow. Passive traffic analysis such as packet counting will fail. Moreover, due to differences in router bandwidth and other network dynamics, the capacity of circuits vary [149, 161]. Some cells with larger sequence numbers from a fast circuit (e.g., cell 4 on the first circuit in Figure 4.4) may arrive earlier than those with smaller sequence numbers but assigned to a slow circuit (e.g., cell 1 on the second circuit). The cells must be reordered at SRP. Consequently, the waiting time introduced by such reordering will distort or destroy the inter-cell timing correlations of a manipulated subflow and makes active traffic analysis less effective.

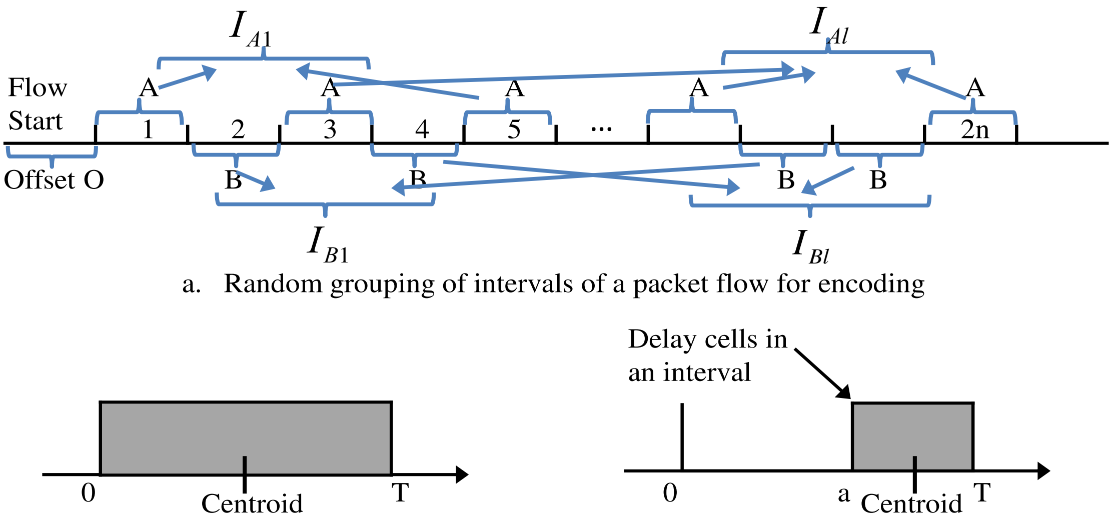
## 4.5 Experiment Evaluation

In this section we test the performance of *m*TorHS-S against a well-known active traffic analysis scheme, i.e., interval centroid-based watermarking (ICBW) [151], and evaluate the enhanced anonymity in our multipath hidden services. In particular, we conducted experiments with the Shadow simulator [88], which is an accurate, discrete event simulator running real Tor protocol over a simulated Internet topology. We implemented the multipath Tor router (please read [161] for details) and plugged it into the Shadow simulator to support multipath hidden services in a private Tor network. We also implemented an adversary node following the threat model described in Section 4.3, which first inserted watermarks to flows at the malicious entry guard OR1 using ICBW protocol, and then examined packets at the malicious client rendezvous point for expected traffic signatures

### 4.5.1 Implementing ICBW watermarking scheme

To assess the resistance of the proposed scheme against traffic analysis, we implemented a state-of-the-art traffic watermarking scheme, the interval centroid-based watermarking (ICBW) protocol to attack low-latency anonymity systems [151]. The ICBW was verified on a leading commercial anonymizing service platform www.anonymizer.com as an effective attack. Here we briefly explain its working mechanism. As illustrated in Figure 4.5, ICBW embeds a watermark into a sufficiently long flow by intentionally changing the centroid of several randomly selected intervals. This scheme divides the duration of flow starting from an offset $O$ into *2n* intervals of equal length $T$. The centroid is then calculated by averaging each packet's relative arrival time to the start of its interval. The intervals are randomly grouped into two subsets ($\{I_{A_1},...,I_{A_l}\}$ and $\{I_{B_1},...,I_{B_l}\}$), each with $l$ elements. Each element in set $A$ and $B$ contains $r$ intervals for redundancy such that $n = rl$. The random grouping is illustrated in Figure 4.5a. To encode a watermarking bit 1 (or 0), two elements ($I_{A_i}$ and $I_{B_j}$) of the set $A$ and $B$ are selected, respectively. The packets in all intervals of $I_{A_i}$ (or $I_{B_i}$) will be delayed by a maximum value of $a$. Figure 4.5b illustrates the delaying, which actually changes the distribution of relative arrival time from $U(0,T)$ to $U(a,T)$ where $U(,)$ stands for uniform distribution. After encoding, the difference between the average centroids of $I_A$ and $I_B$ will be $\frac{a}{2}$ for watermark bit 1 and $-\frac{a}{2}$ for watermark bit 0. To decode, the decoder starts from the same offset $O$ and checks the existence of the watermark. Because each watermark bit is encoded by averaging the delays of packets that are randomly selected from many intervals, ICBW is very robust to network delay jitter and flow mixing along a circuit. We refer the readers to [151] for details.

For ICBW, we follow the suggested parameter setting: 32-bit watermarks are randomly generated and the redundancy $r$ is set to 20. The interval length $T$ and the maximum delay $a$ are set to 500ms and 350ms, respectively. We use an offset $O = 10s$ to delay cells in the selected intervals according to the watermark bits at the malicious guard and meanwhile log the arrival time of each cell using the same offset at CRP. From the logged arrival time we compute the difference between the average centroid of $I_A$ and $I_B$ to derive a watermark bit 1 if it is closer to $\frac{a}{2}$ or 0 if it is closer



a. Random grouping of intervals of a packet flow for encoding

b. Distribution of cell timing before and after delaying

Figure 4.5: Random grouping intervals for a packet flow in ICBW [96, 151]

to $-\frac{a}{2}$. Hamming distance, which is the number of mismatched bits, is computed between the derived watermark and the original watermark to evaluate how successful the watermarking attack is. Since the network delay is unknown, a set of different offsets are tested. The one that matches most to the inserted watermarks is chosen as the correct decoding offset to decode the watermark.

### 4.5.2 Implementing *m*TorHS-S

We implemented *m*TorHS on Tor v0.2.5.6-alpha. The construction of the client's circuit remains the same as in the current Tor hidden service design, but we change the implementation for the server circuit construction. As explained previously, the server's circuit consists of an entry guard, a middle relay, SRP and CRP. Since we assume the malicious client controls the guard node and CRP, we fix the selection of the two nodes in the implementation. The middle relay is randomly selected from the Tor router set and SRP is randomly selected from the "rendezvous guard set". For simplicity, in our current implementation we form the rendezvous guard set with routers flagged as entry guard. This is because the concept of "rendezvous guard set" is derived from the idea of entry guard set to denote a set of routers trusted by the hidden server. In the future, we will develop selection criteria to assist the selection of reliable rendezvous guards.

Finally, we build a small private Tor network in the Shadow simulator with 50 Tor routers, 1 hidden server, 20 general HTTP servers, 1 malicious client and 100 general web clients to run our experiments. Among the 50 routers, two are configured as malicious CRP and OR1. This is a general case for evaluation. Obviously, the fewer the general clients in the network, the more likely the adversary identifies the hidden server. So, we choose an extreme setting for comparison, where the adversary is the only client in the network. As pointed out by Wang et al. in [151], the longer the flow, the more robust the watermark. To ensure a sufficiently long flow for successful watermarking, we let the client to request a 100MiB file at the hidden server under both settings.

### 4.5.3 Results

We perform the ICBW attack on the original Tor and the proposed *m*TorHS-S, where *m* is set to 2, 4, 6 and 8. To rule out random noise, we repeat the watermarking attack for ten times for each setting. The results shown below are the average results of ten experiments. Table 4.2 shows the comparison in terms of Hamming distance between Tor and *m*TorHS-S under different settings. A larger Hamming distance indicates that the anonymity system can better transform the original flow and prevent the traffic analysis. No matter in general cases or extreme cases, *m*TorHS-S can better obscure the embedded watermark in the victim's flow. The Hamming distance achieved on *m*TorHS-S is always larger than the maximum Hamming distance threshold (i.e., 8) [151]. When the Hamming distance exceeds the threshold, the adversary has less confidence to correlate the watermarked flow to the suspected flow. We note that with *m* increasing, the Hamming distance does not increase obviously. One reason might be that when we decode the watermark, we tried a set of different offsets and picked the minimum value.

| | Tor | *m*TorHS-S | | | |
|---|---|---|---|---|---|
| | | m=2 | 4 | 6 | 8 |
| General case | 6 | 9 | 9 | 10 | 12 |
| Extreme case | 3 | 8 | 9 | 9 | 11 |

Table 4.2: Comparison of Hamming distance between Tor and *m*TorHS-S with different *m* where each flow is encoded using different watermarks.

From the adversary's perspective, if she wants to circumvent the multipath routing scheme, she should use the same watermark for different flows. Since two circuits between OR1 and HS multiplex the common TCP connection, the cells that HS sends out through two different subflows arrive at OR1 within the same coding interval of 500ms. If the adversary uses different watermarks to encode them, it is possible that one subflow is delayed while the other one not, which causes the distribution of the delayed subflow is squeezed to $U(a,T)$ while the other one is still $U(0,T)$. When SRP receives these cells from two subflows, SRP will merge them so that the distribution of the merged flow will be a uniform distribution $U(x,T)$ where $0 < x < a$ depending on which subflow the majority of the cells belong to. Therefore, when the malicious CRP receives the merged subflow, she cannot recover the correct centroid of this interval. To avoid this, the adversary ought to use the same watermark to encode all flows going through OR1 so that they will have a same distribution. When they are merged at SRP, the merged flow still preserves the distribution. Table 4.3 shows the results when the adversary embeds the same watermark to all flows. In order to verify this assumption without being influenced by general traffic, we perform this experiment in the extreme cases. As shown in Table 3, when the same number of subflows are used, the Hamming distance of the merged flow in cases where a same watermark is embedded is always smaller than the one when different watermarks are used.

| | Tor | *m*TorHS-S | | | |
|---|---|---|---|---|---|
| | | m=2 | 4 | 6 | 8 |
| Extreme case | 3 | 4 | 7 | 6 | 7 |

Table 4.3: Comparison of Hamming distance between Tor and *m*TorHS-S with different *m* where all flows are encoded using the same watermark.

However, once the adversary encodes multiple flows with the same watermark, her watermarking is vulnerable to the multi-flow attacks [96] (from the defending perspective, we call it multi-flow detection (MFD) in this chapter.) The idea of MFD is that the MFD detector will aggregate all flows into a single flow after it collects a number of watermarked flows. This aggregation scheme in MFD is different from our subflow mixing, which overlaps the relative arrival time of each flow to the same start. If several abnormally long periods of silence (i.e., no packets for hundreds of

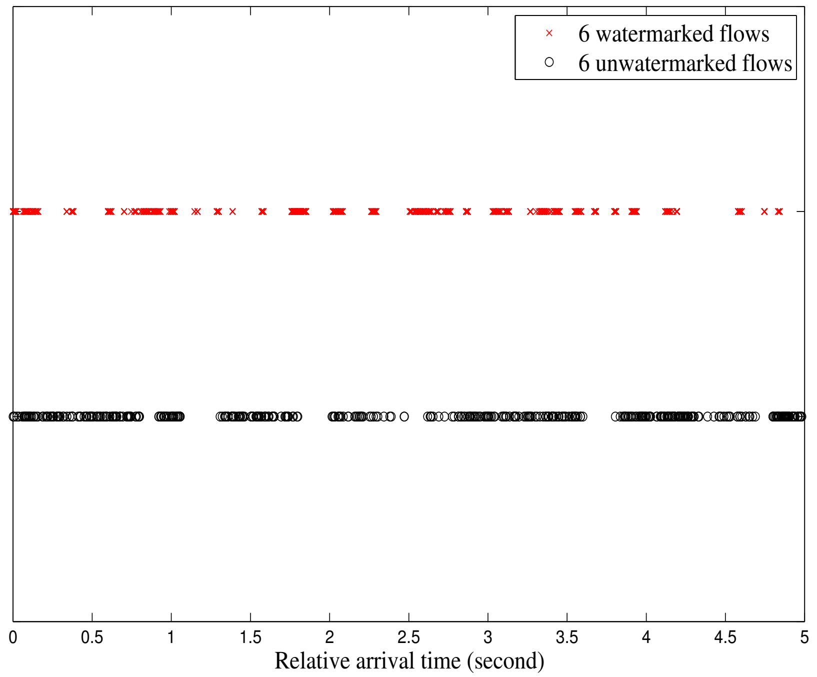


Figure 4.6: Comparison of time pattern of the aggregated flow between watermarked flows and unwatermarked flows.

milliseconds) are observed in the aggregated flow, the detector considers the presence of watermarking attack, extracts the watermarking keys and removes the watermarks from the observed flows. Since SRP merges multiple subflows, which is naturally compatible to MFD, we deploy the MFD detector at SRP. Figure 4.6 shows the aggregated arrival time of six flows with and without the presence of watermark. Compared to the aggregated unwatermarked flows, the silence of the victim's aggregated flow is more obvious and periodic. Once SRP recognizes the existence of a watermark with higher confidence, it can remove it by randomly delaying some cells on the suspicious flow.

## 4.6 Summary

Hidden services based gateway can improve the security of IoT smart home, but it is vulnerable to traffic analysis attacks especially when the entry guard protection is broken. In this chapter, we propose a multipath routing based scheme that exploits flow mixing and flow merging to distort or destroy inserted traffic patterns in a victim's flow. We believe this is an effective complement to the existing protection mechanism. Therefore, our "partial" multipath routing Tor hidden services scheme is suitable for users who require very strong security. Besides, since the multipath architecture is naturally compatible to detection mechanisms based on multiflows, it can be further integrated with multiflow detection protocols to detect the presence of watermarks. We experimentally verify the effectiveness of our scheme in defending one of the most robust watermarking schemes on the Shadow simulator.

# Part II

# Protect Data Privacy of IoT Devices in Data Storage, Utilization and Dissemination

# Chapter 5

# RSPP: A Reliable, Searchable and Privacy-Preserving e-Healthcare System for Cloud-Assisted Body Area Networks

## 5.1 Introduction

In Part I, we explore the solutions of protecting IoT devices from cyberattacks using Tor hidden services. To address the performance and security of Tor hidden services, we present the corresponding solutions for them in Chapter 3 and Chapter 4. In the second part, we will present our first solution to protect data for IoT applications. In this chapter, we particularly focus on how to secure the IoT data storage on cloud and meanwhile keep the usability of data.

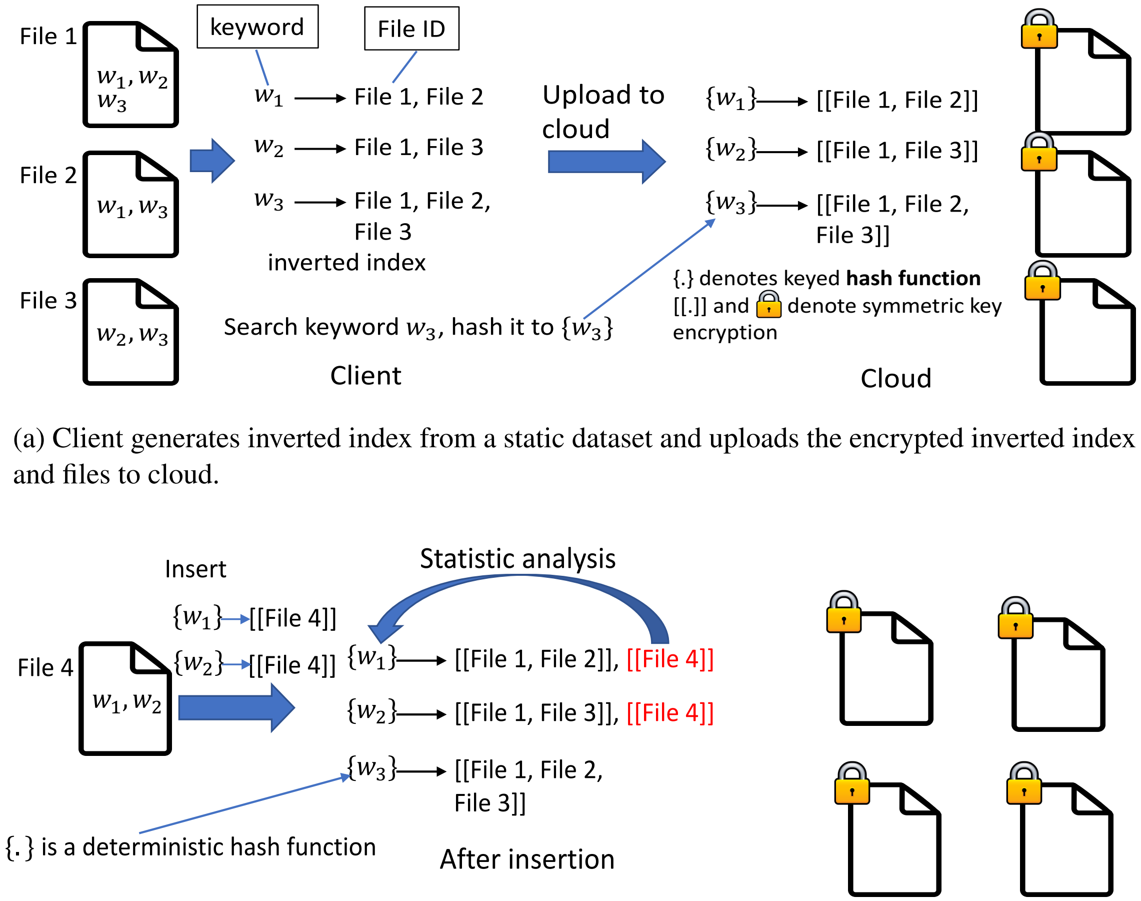
In recent years, with the fast development of cloud computing and Internet of Things (IoT), the conventional healthcare industry is being reshaped to a more flexible and efficient paradigm of e-healthcare. In a typical e-healthcare setting, a group of wearable and/or implantable devices (e.g., smart watches, bracelets, pacemakers, etc.), which forms a wireless body area network (BAN), gathers key vital signs (e.g., heart rate, blood pressure, temperature, pulse oxygen, etc.) from patients at home periodically. Those information is aggregated into a single file called personal health information (PHI) at an IoT gateway and then forwarded to a cloud server for storage. Third-party healthcare service providers (HSPs) can monitor patients' PHI and provide timely diagnosis and reactions by submitting on-demand queries to cloud storage. Although the increasing adoption of cloud computing and IoT services in healthcare industry helps reduce IT cost and improves patient outcomes, security and privacy of PHI are still major concerns as highlighted by the numerous reported data breaches due to malicious attacks, software bugs or accidental errors [97]. In particular, the healthcare regulations such as the Health Insurance Portability and Accountability Act (HIPAA) explicitly require that PHI be secured even as it migrates to the cloud.

While simply encrypting PHI before outsourcing it to the cloud can ensure the regulatory compliance of a healthcare system, it makes PHI utilization (e.g., query by third party HSPs) particularly challenging. Searchable encryption technology (see [134, 56, 41] for pioneering work), which allows encrypted documents to be searched as is by augmenting them with an encrypted search index, provides a promising solution to addressing the aforementioned dilemma. An important line of research on searchable encryption is searchable symmetric encryption (SSE), which is considered more practical in terms of search efficiency for large datasets when compared to its public key-based counterpart. During the past decade, many provably secure SSE schemes [56, 147, 48, 94, 93, 118, 47, 81, 135] have been proposed, which make trade-offs among security, search performance and storage overhead by exploring static- [56, 147, 48] and dynamic datasets [94, 93, 118, 47, 81, 135] as well as various data structures such as an inverted index [56, 94], a document-term matrix [98], a dictionary [47], etc.

We note that previous SSE schemes mainly focus on general search applications on encrypted database. The static SSE schemes that process static datasets and do not support subsequent updates are clearly not suitable for our e-healthcare applications. Moreover, most previous dynamic SSE schemes (except for [81]) work on a setting where a large static dataset is first processed and outsourced to the cloud storage, followed by a number of (infrequent) update operations, which is quite different from the e-healthcare applications where PHI files are created and uploaded to the cloud periodically at a fixed frequency (e.g., every 10 minutes). If we adopt the static SSE scheme for the streaming IoT data, the process of adding new files will lead to a statistical attack. To help interested readers to understand this problem, we show an example in Figure 5.1: The static dataset consists of 3 files as shown in Figure 5.1a. To realize SSE, client first extract keywords from the dataset and builds inverted index for it. Then, he hashes the keyword in the inverted index with a keyed hash function and encrypts the file itself with AES. After that, client uploads them to the cloud. To search for a keyword, e.g., $w_3$, client just hashed it and sends the search token to cloud, and then cloud can look it up and find the corresponding file list. However, if client adds a new file to dataset, statistical attacks happens, which is shown in Figure 5.1b. To add $File_4$ containing keywords $w_1$ and $w_2$, client needs to hash the keyword and encrypt file, and then upload. Since the hash function is deterministic, when cloud appends $File_4$ to the tail of $w_1$, it knows 3 files contains $w_1$. Even though it does not know the content of $w_1$, after many files are uploaded one by one, the cloud can obtain the frequency information of all keywords. We know, in English, some words are used more frequently than others. So, the cloud can compare the obtained frequency to the existing knowledge. Then, the cloud can guess what the keywords are. The process relates to the concept of *forward privacy.* For a dynamic SSE scheme, forward privacy means that when a new keyword and file identifier pair is added, the cloud server does not know anything about this pair [135].

To prevent the cloud server from inferring sensitive information related to a patient (e.g., activity pattern, diet habit, etc.) based solely on observation of the stored encrypted indices, a dynamic SSE scheme with forward privacy is highly desirable. In addition, for remotely monitoring patients' health status, HSPs should be able to perform search on PHIs encrypted by patients. Hence, our system should support a multi-user setting where the data owner and data user might be different. Last but not least, the reliability of an e-healthcare system is also critical and any incorrect or incomplete search results could lead to significant consequences, thereby highlighting the requirement for a verification mechanism to be deployed into the system.

Motivated by the above observations, we present a reliable, searchable and privacy-preserving e-healthcare system for cloud assisted BAN in this paper. The proposed system is built upon a novel dynamic SSE scheme with forward privacy and delegated verifiability, which enables both


File 1
$w_1, w_2$ $w_3$
File 2
$w_1, w_3$
File 3
$w_2, w_3$
keyword
File ID
$w_1$ → File 1, File 2
$w_2$ → File 1, File 3
$w_3$ → File 1, File 2, File 3
inverted index
Upload to cloud
$\{w_1\}$ → [[File 1, File 2]]
$\{w_2\}$ → [[File 1, File 3]]
$\{w_3\}$ → [[File 1, File 2, File 3]]
{.} denotes keyed hash function
[[.]] and 🔒 denote symmetric key encryption
Search keyword $w_3$, hash it to $\{w_3\}$
Client
Cloud
Insert
$\{w_1\}$ → [[File 4]]
$\{w_2\}$ → [[File 4]]
File 4
$w_1, w_2$
Statistic analysis
$\{w_1\}$ → [[File 1, File 2]], [[File 4]]
$\{w_2\}$ → [[File 1, File 3]], [[File 4]]
$\{w_3\}$ → [[File 1, File 2, File 3]]
{.} is a deterministic hash function
After insertion

(a) Client generates inverted index from a static dataset and uploads the encrypted inverted index and files to cloud.

(b) Client adds a new to the existing dataset, which causes statistical attack.

Figure 5.1: SSE scheme and its forward privacy problem for new data.

patients and HSPs to conduct privacy-preserving search on the encrypted PHIs stored in the cloud and verify the correctness and completeness of retrieved search results simultaneously. Our contributions can be summarized as follows:

1. We proposed a dedicated and efficient dynamic SSE scheme for e-healthcare applications where PHIs are generated and stored in the cloud periodically. Our scheme is able to achieve a sub-linear search efficiency and forward privacy by maintaining an increasing counter for each keyword at an IoT gateway.

2. We presented an efficient mechanism that provides patient-controlled search capability for HSPs, thereby extending our system to a multi-user setting. This desired property is realized

through a novel application of Bloom filter [39] on the data owner (i.e., a patient) side.

3. We designed a lightweight delegated verification scheme based on a combination of Bloom filter, Message Authentication Codes (MACs) and aggregate MACs [95], which enables patients to delegate the capability of verifying the search results to HSPs.

## 5.2 Problem Formulation

### 5.2.1 System Model

The system model of our proposed reliable, searchable and privacy-preserving e-healthcare system involves four entities as shown in Fig. 5.2: a patient, an IoT gateway, a cloud server and several HSPs. The patient is the data owner whose health status is monitored by a group of wearable devices forming a BAN. The IoT gateway is the data aggregator which aggregates the periodically collected data into a single PHI file, extracts keywords, builds an encrypted index, and encrypts the PHI files. The encrypted index and PHI files are then sent to the cloud server for storage. Multiple HSPs act as the data users that provide healthcare services for the patient by querying and retrieving his/her encrypted PHIs from the cloud. We note that the e-healthcare system described above has the following unique properties with respect to data processing:

- The PHI files are created by the IoT gateway and stored in the cloud *periodically* (e.g., every 10 minutes).
- The PHI files are *always added* into the cloud storage and file deletion or modification is not needed.
- The total number of unique keywords extracted from all the PHI files is not very large, due to the limited range of values for vital signs.

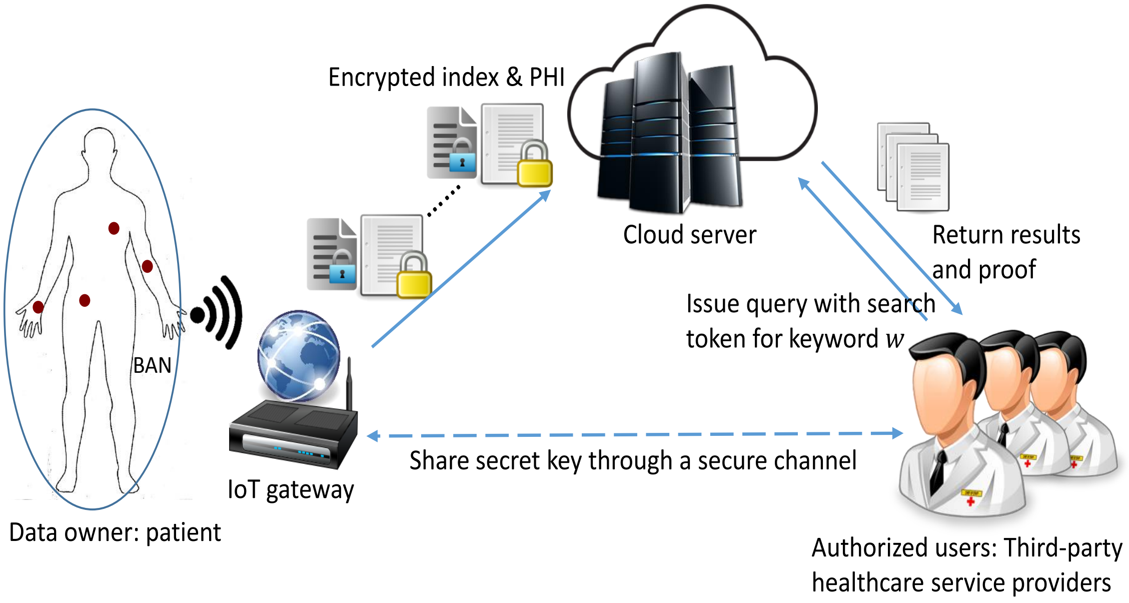


Figure 5.2: The system model of a reliable, searchable and privacy-preserving e-healthcare system.

### 5.2.2 Threat Model

As assumed in most previous work on SSE [134, 56, 94, 135, 81], the cloud server is generally "honest-but-curious", thereby faithfully performing the protocol but making inferences about the stored encrypted documents and data owner's private information. More specifically, in our e-healthcare system, the cloud server might try to infer whether a newly uploaded PHI file contains certain keyword or two PHI files contain the same keyword. Furthermore, the cloud server may also observe the queries submitted by HSPs (so-called search pattern) or the search results (so-called access pattern) to determine whether the same keyword is being searched. Additionally, considering the possibility of accidental system errors on the cloud server as well as the potential attacks from external adversaries, the cloud server might return incorrect or incomplete search results to data user. Finally, we assume that there is no collusion between data users and cloud server, or between users.

### 5.2.3 Design Goals

In this work, we aim to design a reliable, searchable and privacy-preserving e-healthcare system which enables third-party HSPs to provide healthcare services for patients by searching on their encrypted PHIs incrementally stored on the cloud in a privacy-preserving and verifiable manner. The design goals of our system are as follows:

1. **Search efficiency**. The search complexity on the cloud should be optimally sub-linear $O(k)$, where $k$ is the number of PHIs containing the queried keyword.

2. **Forward privacy**. The cloud should not learn whether the newly stored PHIs contain some specific keywords.

3. **Multi-user support**. HSPs should be able to perform patient-controlled search on behalf of a patient.

4. **Verifiability**. HSPs should be able to verify the correctness and completeness of the search results.

Note that hiding search and access patterns in a general SSE setting can be achieved using the oblivious RAM (ORAM) [75]. However, ORAM-based schemes, while providing strong protection for privacy, incur significant computational and communication overhead for search. To ensure the practicality of our system, we did not consider employing the ORAM based approach to protect those patterns in this work.

## 5.3 Notations, Preliminaries and Definition

### 5.3.1 Notations and Preliminaries

Let $e \leftarrow S$ denote selecting an element $e$ from a set $S$ uniformly at random, $\{0,1\}^n$ be the set of binary strings of length $n$, $\{0,1\}^*$ be the set of all finite length binary strings, and $||$ denote the concatenation of two strings. The data file $f$ is uniquely identified by the identity $\mathsf{ID}(f)$ and contains a set of distinct keywords $W(f) = \{w_1,...,w_l\}$. Let $\mathsf{TBL}$ be a hash table storing key-value pairs $(\mathsf{key},\mathsf{val})$ such that $\mathsf{TBL}[\mathsf{key}] = \mathsf{val}$, $\mathsf{TBL}[\mathsf{key}] := \mathsf{val}$ denote assigning $\mathsf{val}$ to $\mathsf{key}$, and $\mathsf{key} \in \mathsf{TBL}$ denote that $\mathsf{key}$ is an element of the key set in $\mathsf{TBL}$.

Let $\mathscr{F}_1 : \{0,1\}^{\lambda} \times \{0,1\}^* \to \{0,1\}^{\lambda}$, $\mathscr{F}_2 : \{0,1\}^{\lambda} \times \{0,1\}^* \to \{0,1\}^{2\lambda}$, $\mathscr{F}_3 : \{0,1\}^{\lambda} \times \{0,1\}^* \to \{0,1\}^{3\lambda}$ be three secure pseudorandom functions and $\mathscr{H} : \{0,1\}^* \to \{0,1\}^{\lambda}$ be a secure hash function. Let $\mathsf{SE} = (\mathsf{SE.GenKey},\mathsf{SE.Enc},\mathsf{SE.Dec})$ be a semantic secure symmetric encryption where $\mathsf{SE.GenKey}$ is the key generation algorithm, $\mathsf{SE.Enc}$ is the encryption algorithm and $\mathsf{SE.Dec}$ is the decryption algorithm. Let $\mathsf{Mac} = (\mathsf{Mac.GenKey},\mathsf{Mac.GenMac})$ be a secure message authentication code scheme, where $\mathsf{Mac.GenKey}$ is the key generation algorithm, and $\mathsf{Mac.GenMac}$ is the message authentication code generation algorithm.

Bloom filter is a space efficient data structure to represent a set $S$ and allow efficient membership query. A Bloom filter $\mathsf{BF}$ is an array of $m$-bit, which are set to 0 initially, and associated with $k$ independent universal hash functions $\mathscr{H}_1, \ldots, \mathscr{H}_k$, such that $\mathscr{H}_i : \{0,1\}^* \to \{0,\ldots,m-1\}$. Given $e \in S$, the bits with respect to $\mathscr{H}_i(e), 1 \leq i \leq k$, are set to 1. To query whether $e$ is an element of $S$ or not, one can check whether all bits with respect to $\mathscr{H}_i(e), 1 \leq i \leq k$, are equal to 1. If not, $e \notin S$ for sure. Otherwise, $e \in S$ in a high probability due to the false positive rate. Suppose the outputs of all hash functions are in uniform random distribution and $n$ elements are hashed into the $\mathsf{BF}$, the false positive rate is $(1-e^{-kn/m})^k$. A $\mathsf{BF}$ usually associates with two algorithms:

- $\mathsf{BF} \leftarrow \mathsf{BFAdd}(\mathsf{BF}, e)$ : This algorithm hashes an element $e$ into the Bloom filter $\mathsf{BF}$.
- $\{0,1\} \leftarrow \mathsf{BFVerify}(\mathsf{BF}, e)$: This algorithm outputs 1 if $e$ is an element of $S$ where all elements were hashed into $\mathsf{BF}$ (with certain false positive rate); and 0 otherwise.

### 5.3.2 Definition for Dynamic Symmetric Searchable Encryption

Similar to the notation [135], let $((c_{out}),(s_{out})) \leftarrow protocol((c_{in}),(s_{in}))$ denote the protocol running between the data owner and the server, where the data owner takes as input $c_{in}$ and outputs $c_{out}$, and the server takes as input $s_{in}$ and outputs $s_{out}$.

**Definition 1** *The verifiable* DSSE *scheme that supports streaming data consists of the following algorithms/protocols:*

- $\mathsf{K} \leftarrow \mathsf{GenKey}(1^{\lambda})$*: Given a security parameter* $\lambda$*, the data owner runs the algorithm to generate the secret key* $\mathsf{K}$.
- $((\mathsf{state}'_c),(\mathsf{state}'_s,C)) \leftarrow \mathsf{AddFile}((\mathsf{K},\mathsf{state}_c,f),\ (\mathsf{state}_s))$*: The data owner takes as inputs the secret key* $\mathsf{K}$*, current state information* $\mathsf{state}_c$ *and the file* $f$ *containing a set of keywords* $W(f)$*, and the server takes as input its current state information* $\mathsf{state}_s$*. The data owner runs this protocol to outsource* $C$ *(the encryption form of the file* $f$*) to the server and updates its own state to* $\mathsf{state}'_c$*. The server also updates its own state to* $\mathsf{state}'_s$*. Initially, both* $\mathsf{state}_c$ *and* $\mathsf{state}_s$ *are empty.*
- $\mathsf{token} \leftarrow \mathsf{GenToken}(\mathsf{K},\mathsf{state}_c,w)$ : *The data owner runs this algorithm to generate search token* $\mathsf{token}$*, by taking as input* $\mathsf{K}$*,* $\mathsf{state}_c$ *and* $w$*.*
- $(\mathsf{rst},\mathsf{prof}) \leftarrow \mathsf{Search}(\mathsf{state}_s,\mathsf{token})$*: Given the search token* $\mathsf{token}$*, the server runs this algorithm to output the search result* $\mathsf{rst}$ *consisting of a set of file identifiers. Moreover, the server generates the proof* $\mathsf{prof}$ *showing the correctness of the search result.*
- $\{0,1\} \leftarrow \mathsf{SSEVerify}(\mathsf{K},\mathsf{state}_c,w,\mathsf{rst},\mathsf{prof})$*: The data owner (or authorized user) runs this algorithm to verify the correctness of the search result* $\mathsf{rst}$*, given* $\mathsf{K}$*,* $\mathsf{state}_c$*,* $w$*, and* $\mathsf{prof}$.

Basically, the verifiable DSSE scheme supporting streaming data aims to achieve the following security goals: forward privacy, verifiability and confidentiality of outsourced data and queried keyword.

## 5.4 Dynamic SSE Achieving Forward Privacy

For the sake of simplicity, we first present the DSSE construction achieving forward privacy, and leave the full-fledged DSSE design to the next section.

### 5.4.1 Design Rational

Informally, forward privacy in $\mathsf{DSSE}$ demands that when adding a new file, the server should not learn whether the newly added file contains certain keyword that has been queried before or not, unless the keyword is queried again. Therefore, it is sufficient to achieve forward privacy if any keyword in the newly added file will not be linked to any encrypted keywords stored in the server.

Instead of using computationally heavy cryptographic primitives (e.g., ORAM), in this paper we exploit the combination of locally stored state information and chaining technique in a subtle way, and utilize the lightweight cryptographic primitives to achieve forward privacy, which is explained as follows.

The data owner associates to each keyword a counter, indicating the number of outsourced encrypted files having the keyword so far. That is, the data owner locally maintains the state information (i.e., pairs of keyword and counter). Suppose the counter associated to keyword $w$ is $\mathsf{cnt}$, the index with respect to $w$, stored in the server, is a collection of tuples $\{(\tau_1, \mathsf{ID}(f_1)), \ldots, (\tau_{\mathsf{cnt}}, \mathsf{ID}(f_{\mathsf{cnt}}))\}$ where $\tau_i = \mathscr{F}_1(K, w||i), 1 \leq i \leq \mathsf{cnt}, \mathscr{F}_1$ is a secure pseudorandom function, $K$ is a private key and $f_1, \ldots, f_{\mathsf{cnt}}$ are files having keyword $w$. When adding a new file $f$ containing the keyword $w$, the data owner sends to the server the following tuple

$$(\tau_{\mathsf{cnt}+1},\ \mathsf{ID}(f))$$

where $\tau_{\mathsf{cnt}+1} = \mathscr{F}_1(K, w||\mathsf{cnt}+1)$. Thanks to $\mathscr{F}_1$, without knowing $K$ the server cannot know whether $\tau_{\mathsf{cnt}+1}$ is generated from the same keyword as that of $\tau_i, 1 \leq i \leq \mathsf{cnt}$. Note that the data owner does not need to maintain all previous states for each keyword because file deletion is not needed in healthcare.

While binding counter to a keyword can break the correlation of two identical keywords, it raises another challenge: given one search token generated from the keyword and the counter, the server can only retrieve one single file identifier. That is, to retrieve all file identifiers having the specific keyword, the data owner has to enumerate all previous counters and generate search tokens, which is rather costly in term of bandwidth for search.

To mitigate this disadvantage, we use the following chaining technique, which implicitly links the tuples corresponding to the same keyword together (let $\tau_i = \mathscr{F}_1(K,w||i), 0 \leq i \leq \mathsf{cnt}$):

$$\begin{array}{ccc}
\tau_1, & \langle \tau_0 || 0^\lambda \rangle \bigoplus \mathscr{F}_2(K_1, \tau_1), & \mathsf{ID}(f_1) \\
\tau_2, & \langle \tau_1 || K_1 \rangle \bigoplus \mathscr{F}_2(K_2, \tau_2), & \mathsf{ID}(f_2) \\
 & \cdots & \\
\tau_{\mathsf{cnt}}, & \langle \tau_{\mathsf{cnt}-1} || K_{\mathsf{cnt}-1} \rangle \bigoplus \mathscr{F}_2(K_{\mathsf{cnt}}, \tau_{\mathsf{cnt}}), & \mathsf{ID}(f_{\mathsf{cnt}})
\end{array}$$

where $\mathscr{F}_2$ is another secure pseudorandom function and $K_i, 1 \leq i \leq \mathsf{cnt}$, is a random key derived from the counter $i$. Obviously, without knowing $K_i, i \geq \mathsf{cnt}$, the server cannot correlate $\tau_{\mathsf{cnt}}$ with $\tau_j, j < \mathsf{cnt}$, even though they might be generated from the same keyword (but different counter). On the other hand, given $\tau_{\mathsf{cnt}}$ and $K_{\mathsf{cnt}}$, the server is able to obtain $\mathsf{ID}(f_{\mathsf{cnt}})$ and recover $\tau_{\mathsf{cnt}-1}$ and $K_{\mathsf{cnt}-1}$ by computing

$$\langle \tau_{\mathsf{cnt}-1} || K_{\mathsf{cnt}-1} \rangle \bigoplus \mathscr{F}_2(K_{\mathsf{cnt}}, \tau_{\mathsf{cnt}}) \bigoplus \mathscr{F}_2(K_{\mathsf{cnt}}, \tau_{\mathsf{cnt}}).$$

The server then obtains all file identifiers by iterating such process until that the key is $\lambda$-bit of zero.

A simplified example is shown in Figure 5.3 to illustrate the rationale. In Figure 5.3a, client first add $File_1$ which contains 3 keywords, so 3 entries are inserted to the local table where $ctr$ is the counter for the corresponding keyword. Then, clients uploads the inverted index to the cloud. We can see that, on the cloud side in Figure 5.3a, the key of the remote table is in the format of keyed hash value, while the value of the table is a virtual address. In Figure 5.3b, to add $File_2$, client first update his local table. Since $File_2$ also contains $w_1$ and $w_3$, we can see their counters are increased to 2. On the cloud side, two new entries are inserted: one for $w_1$ and another for $w_3$. After insertion, although those two entries with gray background contain the same keyword $w_3$, the cloud cannot correlate them because the counters are different. If we take a closer look at the virtual address, we can see that the virtual address points to the previous entry which contains the same keyword. All entries containing the same keyword form a linked list, but this link is invisible to the cloud, because we use the second part to blind the virtual link relation. The relation will be revealed when we search for the keyword. In Figure 5.3c, to search for keyword $w_3$, client can use the keyword and counter to generate a search token. Then, cloud uses the search token to find the entry. After cloud removes the blinder using the key in search token, it can use the virtual address to look up all the previous entries containing $w_3$ until a virtual address with all 0s.

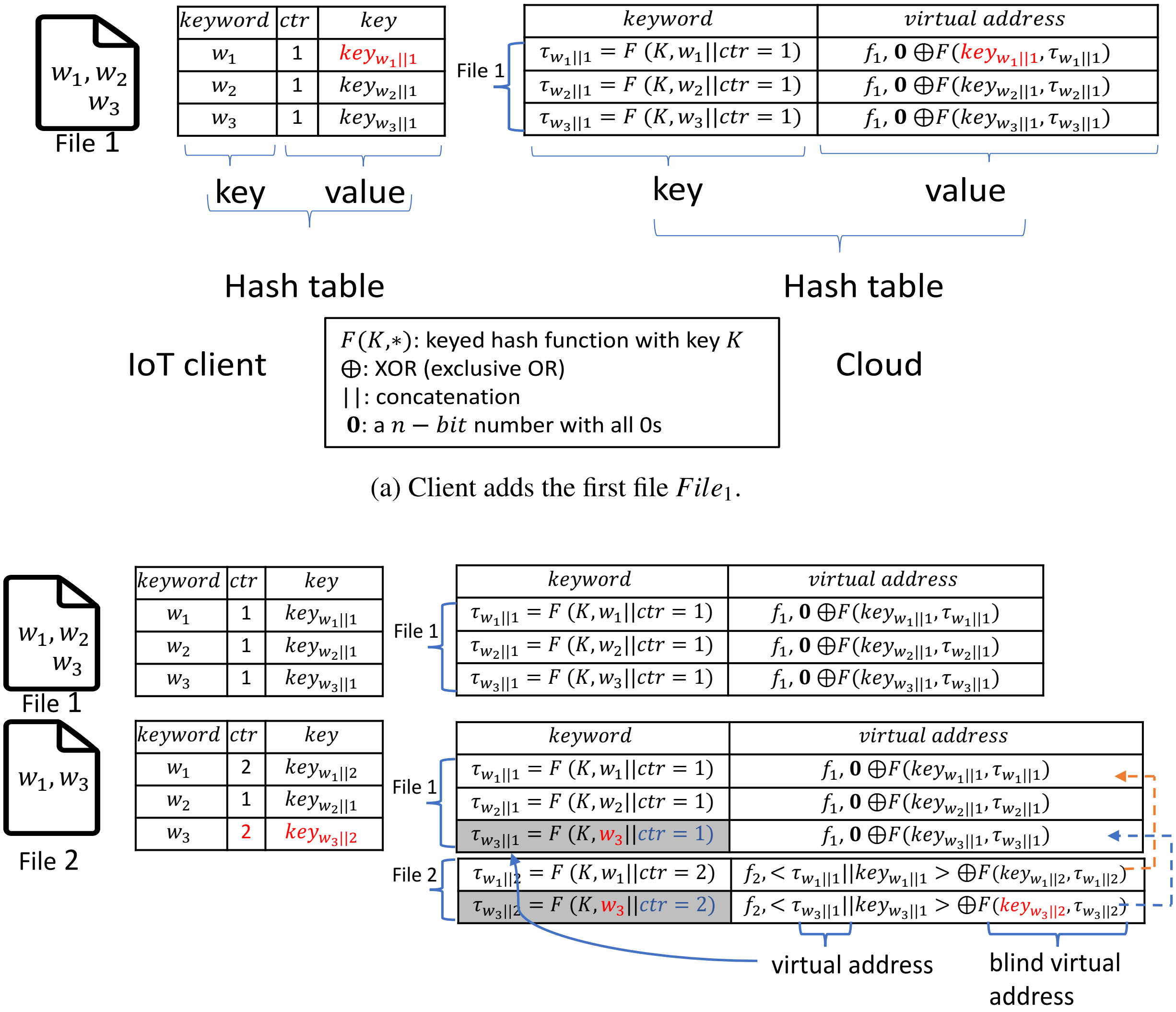
### 5.4.2 Construction

We show the construction in Fig. 5.4. Here the random key $K_{\mathsf{cnt}}$ for keyword $w$ is generated by applying the pseudorandom function such that $K_{\mathsf{cnt}} = \mathscr{F}_1(K, \mathscr{H}(w||\mathsf{cnt}))$. In addition, the data owner stores the state information (i.e., pairs of $(w, \mathsf{cnt})$) in the hash table $\mathsf{TBL}_c$, which maps keyword $w$ to the counter $\mathsf{cnt}$. On the other hand, the server also stores the state information (i.e., the encrypted index) in the hash table $\mathsf{TBL}_s$. We can see that given the keyword $w$, the search complexity is linear to the number of files containing $w$, which is sublinear to the number of outsourced encrypted files.

**Optimization I: Speed up search operation.** Note that the server might be able to speed up the search further: given $\mathsf{token} = (\tau_{\mathsf{cnt}}, K_{\mathsf{cnt}})$ where $\tau_{\mathsf{cnt}} = \mathscr{F}_1(K, w||\mathsf{cnt})$, the server can update its state information by setting $\mathsf{TBL}_s[\tau_{\mathsf{cnt}}] = \perp ||\mathsf{rst}$, where $\perp$ is a stop sign and $\mathsf{rst}$ is the search result with respect to $\mathsf{token}$. By doing this, the server not only accelerates the search without repeating the iterations, but also saves the storage by storing file identifiers only.

## 5.5 Full-fledged DSSE Construction

In this section, we present the full-fledged $\mathsf{DSSE}$. In contrast to the $\mathsf{DSSE}$ presented above, the full-fledged $\mathsf{DSSE}$ not only achieves forward privacy, but also supports search capability enforce-



$w_1, w_2$ $w_3$
File 1
key
value
Hash table
key
value
Hash table
IoT client
Cloud
$F(K,*)$: keyed hash function with key $K$
$\oplus$: XOR (exclusive OR)
| |: concatenation
$\mathbf{0}$: a $n-bit$ number with all 0s

| $keyword$ | $ctr$ | $key$ |
|---|---|---|
| $w_1$ | 1 | $key_{w_1\|\|1}$ |
| $w_2$ | 1 | $key_{w_2\|\|1}$ |
| $w_3$ | 1 | $key_{w_3\|\|1}$ |

| | $keyword$ | $virtual\ address$ |
|---|---|---|
| File 1 | $\tau_{w_1\|\|1} = F\ (K, w_1\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_1\|\|1}, \tau_{w_1\|\|1})$ |
| | $\tau_{w_2\|\|1} = F\ (K, w_2\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_2\|\|1}, \tau_{w_2\|\|1})$ |
| | $\tau_{w_3\|\|1} = F\ (K, w_3\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_3\|\|1}, \tau_{w_3\|\|1})$ |

(a) Client adds the first file $File_1$.

| $keyword$ | $ctr$ | $key$ |
|---|---|---|
| $w_1$ | 1 | $key_{w_1\|\|1}$ |
| $w_2$ | 1 | $key_{w_2\|\|1}$ |
| $w_3$ | 1 | $key_{w_3\|\|1}$ |

| | $keyword$ | $virtual\ address$ |
|---|---|---|
| File 1 | $\tau_{w_1\|\|1} = F\ (K, w_1\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_1\|\|1}, \tau_{w_1\|\|1})$ |
| | $\tau_{w_2\|\|1} = F\ (K, w_2\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_2\|\|1}, \tau_{w_2\|\|1})$ |
| | $\tau_{w_3\|\|1} = F\ (K, w_3\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_3\|\|1}, \tau_{w_3\|\|1})$ |

| $keyword$ | $ctr$ | $key$ |
|---|---|---|
| $w_1$ | 2 | $key_{w_1\|\|2}$ |
| $w_2$ | 1 | $key_{w_2\|\|1}$ |
| $w_3$ | 2 | $key_{w_3\|\|2}$ |

| | $keyword$ | $virtual\ address$ |
|---|---|---|
| File 1 | $\tau_{w_1\|\|1} = F\ (K, w_1\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_1\|\|1}, \tau_{w_1\|\|1})$ |
| | $\tau_{w_2\|\|1} = F\ (K, w_2\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_2\|\|1}, \tau_{w_2\|\|1})$ |
| | $\tau_{w_3\|\|1} = F\ (K, w_3\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_3\|\|1}, \tau_{w_3\|\|1})$ |
| File 2 | $\tau_{w_1\|\|2} = F\ (K, w_1\|\|ctr = 2)$ | $f_2, < \tau_{w_1\|\|1}\|\|key_{w_1\|\|1} > \oplus F(key_{w_1\|\|2}, \tau_{w_1\|\|2})$ |
| | $\tau_{w_3\|\|2} = F\ (K, w_3\|\|ctr = 2)$ | $f_2, < \tau_{w_3\|\|1}\|\|key_{w_3\|\|1} > \oplus F(key_{w_3\|\|2}, \tau_{w_3\|\|2})$ |

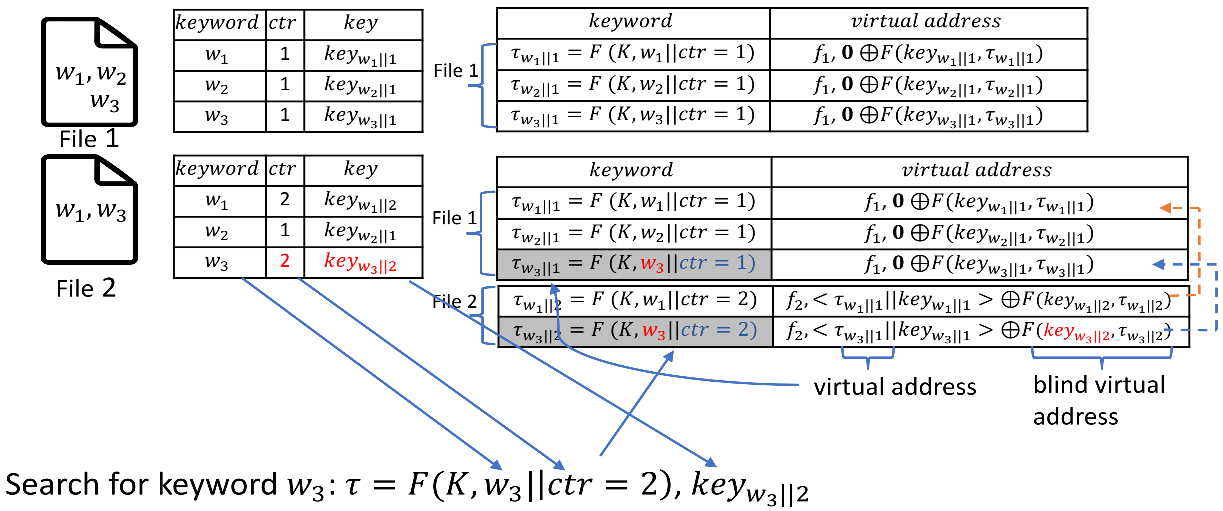
(b) Client adds the second file $File_2$.

Figure 5.3: SSE scheme and its forward privacy problem for new data.

ment and delegated verifiability, where the former allows the data owner (i.e., patients) to enforce controlled search capability, and the latter enables authorized data users (i.e., HSPs) to verify the correctness of the search result.

### 5.5.1 High Level Idea

**Search Capability Enforcement**. In order to enforce search capability, we need to resolve two questions: (i) how to grant authorized data users with search capability; (ii) how to revoke au-



| *keyword* | *ctr* | *key* |
|---|---|---|
| $w_1$ | 1 | $key_{w_1\|\|1}$ |
| $w_2$ | 1 | $key_{w_2\|\|1}$ |
| $w_3$ | 1 | $key_{w_3\|\|1}$ |

| *keyword* | *virtual address* |
|---|---|
| $\tau_{w_1\|\|1} = F\ (K, w_1\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_1\|\|1}, \tau_{w_1\|\|1})$ |
| $\tau_{w_2\|\|1} = F\ (K, w_2\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_2\|\|1}, \tau_{w_2\|\|1})$ |
| $\tau_{w_3\|\|1} = F\ (K, w_3\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_3\|\|1}, \tau_{w_3\|\|1})$ |

| *keyword* | *ctr* | *key* |
|---|---|---|
| $w_1$ | 2 | $key_{w_1\|\|2}$ |
| $w_2$ | 1 | $key_{w_2\|\|1}$ |
| $w_3$ | 2 | $key_{w_3\|\|2}$ |

| *keyword* | *virtual address* |
|---|---|
| $\tau_{w_1\|\|1} = F\ (K, w_1\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_1\|\|1}, \tau_{w_1\|\|1})$ |
| $\tau_{w_2\|\|1} = F\ (K, w_2\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_2\|\|1}, \tau_{w_2\|\|1})$ |
| $\tau_{w_3\|\|1} = F\ (K, w_3\|\|ctr = 1)$ | $f_1, \mathbf{0} \oplus F(key_{w_3\|\|1}, \tau_{w_3\|\|1})$ |
| $\tau_{w_1\|\|2} = F\ (K, w_1\|\|ctr = 2)$ | $f_2, < \tau_{w_1\|\|1}\|\|key_{w_1\|\|1} > \oplus F(key_{w_1\|\|2}, \tau_{w_1\|\|2})$ |
| $\tau_{w_3\|\|2} = F\ (K, w_3\|\|ctr = 2)$ | $f_2, < \tau_{w_3\|\|1}\|\|key_{w_3\|\|1} > \oplus F(key_{w_3\|\|2}, \tau_{w_3\|\|2})$ |

(c) Client searches for keyword $w_3$.

Figure 5.3 (Continued): A simplified example of our basic construction.

thorized data user's privilege if necessary. Furthermore, we require that the approach should be efficient without extensive interaction between the data owner and authorized data users.

Granting search capability requires the data owner to distribute the secret key (i.e., $K_{\mathsf{SE}}$ and $K$) and state information (i.e., the counter for each keyword) to authorized data users efficiently and securely. While secret key distribution can be done efficiently through a one-time off-line setup, state information distribution might be costly because authentication (between the data owner and the authorized user) is needed when authorized data users fetch the fresh state information, which is frequently updated. Note that making the data owner's state information public (even if encrypted) will harm the forward privacy because the server can infer which keyword (or encrypted keyword) was contained in the newly added file.

To address the above issue, we adopt the "document-and-guess" approach: The server maintains a Bloom filter $\mathsf{BF}_s$, and puts each received encrypted keyword $\mathscr{F}_1(K, w||\mathsf{cnt})$ into the Bloom filter $\mathsf{BF}_s$, and the authorized user, having the secret key already and fetching $\mathsf{BF}_s$ from the server, can guess the latest counter value by enumerating $(1, \ldots, \mathsf{cnt}, \mathsf{cnt}+1)$ such that $\mathscr{F}_1(K, w||\mathsf{cnt})$ is an element hashed to $\mathsf{BF}_s$ but $\mathscr{F}_1(K, w||\mathsf{cnt}+1)$ not (suppose the false positive rate of $\mathsf{BF}_s$ is extremely low, e.g., $2^{-30}$ in our experiments).

- $\mathsf{K} \leftarrow \mathsf{GenKey}(1^\lambda)$: Let $\mathscr{F}_1 : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda, \mathscr{F}_2 : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^{2\lambda}$ be two pseudorandom functions, $\mathscr{H} : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a secure hash function and $\mathsf{SE}$ be a secure symmetric key encryption. Given the security parameter $\lambda$, the data owner selects $K \leftarrow \{0,1\}^\lambda$, runs $\mathsf{SE.GenKey}$ to get $K_{\mathsf{SE}}$, and sets $\mathsf{K} = (K_{\mathsf{SE}}, K)$.

- $((\mathsf{state}'_c), (\mathsf{state}'_s, C)) \leftarrow \mathsf{AddFile}((\mathsf{K}, \mathsf{state}_c, f), (\mathsf{state}_s))$: Suppose that the identifier of file $f$ is $\mathsf{ID}(f)$ and the set of keywords extracted from $f$ is $W(f) = \{w_1, \ldots, w_l\}$. Note that when the system was initialized, $\mathsf{state}_c = \mathsf{TBL}_c = \emptyset$ and $\mathsf{state}_s = \mathsf{TBL}_s = \emptyset$ where $\mathsf{TBL}_c$ and $\mathsf{TBL}_c$ are hash tables. The protocol proceeds as follows:

  **The data owner:**

  ```
  Let Ind be an empty set, and run $C \leftarrow \mathsf{SE.Enc}(K_{\mathsf{SE}}, f)$ for file $f$
  for each keyword $w \in W(f)$ do
      Let $K_{\mathsf{prev}} = 0^\lambda$, $\mathsf{cnt} = 1$ and $\mathsf{cnt}_{\mathsf{prev}} = 0$
      if $w \in \mathsf{TBL}_c$ then
          Retrieve cnt from $\mathsf{TBL}_c$ with respect to $w$
          Let $K_{\mathsf{prev}} = \mathscr{F}_1(K, \mathscr{H}(w||\mathsf{cnt}))$, $\mathsf{cnt}_{\mathsf{prev}} = \mathsf{cnt}$ and $\mathsf{cnt} = \mathsf{cnt} + 1$
      end
      Compute $K_{\mathsf{cnt}} \leftarrow \mathscr{F}_1(K, \mathscr{H}(w||\mathsf{cnt}))$
      Compute $\tau_{\mathsf{cnt}} = \mathscr{F}_1(K, w||\mathsf{cnt})$ and
      $\mu_{\mathsf{cnt}} = \langle \mathscr{F}_1(K, w||\mathsf{cnt}_{\mathsf{prev}})||K_{\mathsf{prev}} \rangle \bigoplus \mathscr{F}_2(K_{\mathsf{cnt}}, \tau_{\mathsf{cnt}})$
      Let $\mathsf{TBL}_c[w] := \mathsf{cnt}$ and $\mathsf{Ind} = \mathsf{Ind} \bigcup \{(\tau_{\mathsf{cnt}}, \mu_{\mathsf{cnt}})\}$
  end
  Send $(C, \mathsf{ID}(f), \mathsf{Ind})$ to the server and let $\mathsf{state}'_c = \mathsf{TBL}_c$
  ```

  **The server:**

  Upon receiving $(C, \mathsf{ID}(f), \mathsf{Ind})$ from the data owner, the server proceeds as follows:

  ```
  for each $(\tau, \mu) \in \mathsf{Ind}$ do
      Let $\mathsf{TBL}_s[\tau] := \mu||\mathsf{ID}(f)$
  end
  Store $C$ locally and set $\mathsf{state}'_s = \mathsf{TBL}_s$
  ```

- $\mathsf{token} \leftarrow \mathsf{GenToken}(\mathsf{K}, \mathsf{state}_c, w)$: Given the keyword $w$ to be queried, the data owner generates the search token as follows: (i) Retrieve cnt from $\mathsf{state}_c$ with respect to $w$, (ii) Compute $K_{\mathsf{cnt}} = \mathscr{F}_1(K, \mathscr{H}(w||\mathsf{cnt}))$ and (iii) Let $\mathsf{token} = (\mathscr{F}_1(K, w||\mathsf{cnt}), K_{\mathsf{cnt}})$, which will be sent to the server.

Figure 5.4: The DSSE construction achieving forward privacy. Note that downloaded encrypted files can be decrypted with $K_{\mathsf{SE}}$.

- $\underline{\mathsf{rst} \leftarrow \mathsf{Search}(\mathsf{state}_s, \mathsf{token})}$: Given $\mathsf{token} = (\mathscr{F}_1(K, w||\mathsf{cnt}), K_{\mathsf{cnt}})$, the server conducts the search by letting rst be an empty set, $\tau' = \mathscr{F}_1(K, w||\mathsf{cnt})$, $K' = K_{\mathsf{cnt}}$, and running the following algorithm:

```
while $K' \neq 0^\lambda$ do
    Retrieve $\mu||\mathsf{ID}(f)$ from $\mathsf{TBL}_s$ with respect to $\tau'$ and let $\mathsf{rst} = \mathsf{rst} \bigcup \{\mathsf{ID}(f)\}$
    Let $\tau'||K' = \mu \bigoplus \mathscr{F}_2(K', \tau')$ (which results in $\mathscr{F}_1(K, w||(i-1))||K_{i-1}$ if the current counter is $i$)
end
Return rst as the search result
```

Figure 5.4 (Continued): The DSSE construction achieving forward privacy. Note that downloaded encrypted files can be decrypted with $K_{\mathsf{SE}}$.

On the other hand, in order to allow the data owner to revoke authorized users' search capability, we use the group key idea: The data owner generates a symmetric key $r$, which is securely shared with the server and all authorized users, such that the search token of keyword $w$ generated by authorized users should be $\mathsf{SE.Enc}(r, \mathscr{F}_1(K, w||\mathsf{cnt})||K_{\mathsf{cnt}})$ and the server can recover $(\mathscr{F}_1(K, w||\mathsf{cnt}), K_{\mathsf{cnt}})$ with the stored $r$ via SE.Dec, where SE is a secure symmetric encryption. When an authorized data user was revoked, the data owner only needs to update the group key $r$ to $r'$ and the revoked user cannot generate valid search token without knowing $r'$.

**Delegated Verifiability**. The purpose of delegated verifiability is to allow authorized users (including the data owner) to verify that (i) correctness and completeness of search result, meaning the search result correctly consists of all file identifiers; and (ii) the integrity of the retrieved data files.

First, authorized users can leverage the counter value (if existing) to check whether the server returned the correct number of file identifiers because the counter value indicates the number of files having the specific keyword. Hence, in order to assure that authorized users get correct counter value (which is guessed from $\mathsf{BF}_s$), we need to enable the data user to verify that the cloud faithfully inserts the keywords into Bloom filer. To do so, the data owner also maintains a Bloom filter $\mathsf{BF}_c$, which is built from $\mathscr{F}_1(K, w||\mathsf{cnt})$, and generates a MAC on $\mathsf{BF}_c$ (together with a time stamp). If

- $\underline{\mathsf{K} \leftarrow \mathsf{GenKey}(1^\lambda)}$: Let $\mathscr{F}_1 : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda, \mathscr{F}_3 : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^{3\lambda}$ be two pseudorandom functions, $\mathscr{H} : \{0,1\}^* \rightarrow \{0,1\}^{2\lambda}$ be a secure hash function, SE be a secure symmetric key encryption, Mac be a secure message authentication code. Given the security parameter $\lambda$, the data owner selects $K \leftarrow \{0,1\}^\lambda$, runs $\mathsf{SE.GenKey}$ to get $K_{\mathsf{SE}}$, runs $\mathsf{Mac.GenKey}$ to get $K_{\mathsf{Mac}}$, and sets $\mathsf{K} = (K, K_{\mathsf{SE}}, K_{\mathsf{Mac}})$.

- $\underline{((\mathsf{state}'_c), (\mathsf{state}'_s, C)) \leftarrow \mathsf{AddFile}((\mathsf{K}, \mathsf{state}_c, f), (\mathsf{state}_s))}$ : Suppose that the identifier of file $f$ is $\mathsf{ID}(f)$ and the set of keywords extracted from $f$ is $W(f) = \{w_1, \ldots, w_l\}$. Note that when the system was initialized, $\mathsf{state}_c = (\mathsf{TBL}_c = \emptyset, \mathsf{BF}_c = \emptyset)$ and $\mathsf{state}_s = (\mathsf{TBL}_s = \emptyset, \mathsf{BF}_s = \emptyset)$ where $\mathsf{TBL}_c$ and $\mathsf{TBL}_s$ are two hash tables, and $\mathsf{BF}_c$ and $\mathsf{BF}_s$ are two Bloom filters. The protocol proceeds as follows:

  **The data owner:**

  ```
  Let $\mathsf{Ind}$ be an empty set, and run $C \leftarrow \mathsf{SE.Enc}(K_{\mathsf{SE}}, f)$ for file $f$
  for each keyword $w \in W(f)$ do
      Let $K_{\mathsf{prev}} = 0^\lambda$, $\mathsf{cnt}_{\mathsf{prev}} = 0$, $\mathsf{cnt} = 1$, $\gamma_{\mathsf{prev}} = 0^\lambda$      ($\gamma_{\mathsf{prev}}$ is an aggregate MAC)
      if $w \in \mathsf{TBL}_c$ then
          Retrieve $(\mathsf{cnt}, \gamma_{\mathsf{cnt}})$ from $\mathsf{TBL}_c$ with respect to $w$
          Let $K_{\mathsf{prev}} = \mathscr{F}_1(K, \mathscr{H}(w||\mathsf{cnt}))$, $\mathsf{cnt}_{\mathsf{prev}} = \mathsf{cnt}$, $\gamma_{\mathsf{prev}} = \gamma_{\mathsf{cnt}}$, and $\mathsf{cnt} = \mathsf{cnt} + 1$
      end
      Compute $K_{\mathsf{cnt}} \leftarrow \mathscr{F}_1(K, \mathscr{H}(w||\mathsf{cnt}))$, $\gamma_{\mathsf{cnt}} = \gamma_{\mathsf{prev}} \bigoplus \mathsf{Mac.GenMac}(K_{\mathsf{Mac}}, C||w)$
      (The output of $\mathsf{Mac.GenMac}$ is $\lambda$-bit length)
      Compute $\tau_{\mathsf{cnt}} = \mathscr{F}_1(K, w||\mathsf{cnt})$,
      $\mu_{\mathsf{cnt}} = \langle \mathscr{F}_1(K, w||\mathsf{cnt}_{\mathsf{prev}})||K_{\mathsf{prev}}||\gamma_{\mathsf{cnt}} \rangle \bigoplus \mathscr{F}_3(K_{\mathsf{cnt}}, \tau_{\mathsf{cnt}})$
      Compute $\mathsf{BF}_c \leftarrow \mathsf{BFAdd}(\mathsf{BF}_c, \tau_{\mathsf{cnt}})$
      Let $\mathsf{TBL}_c[w] := (\mathsf{cnt}, \gamma_{\mathsf{cnt}})$, $\mathsf{Ind} = \mathsf{Ind} \bigcup \{(\tau_{\mathsf{cnt}}, \mu_{\mathsf{cnt}})\}$
  end
  Generate the MAC $\sigma \leftarrow \mathsf{Mac.GenMac}(K_{\mathsf{Mac}}, \mathsf{BF}_c||T)$ where $T$ is the current time stamp
  Send $(C, \mathsf{ID}(f), \mathsf{Ind}, \sigma, T)$ to the server and let $\mathsf{state}'_c = (\mathsf{TBL}_c, \mathsf{BF}_c)$
  ```

  **The server:**

  Upon receiving $(C, \mathsf{ID}(f), \mathsf{Ind}, \sigma, T)$ from the data owner, the server proceeds as follows:

  ```
  for each $(\tau, \mu) \in \mathsf{Ind}$ do
      Let $\mathsf{TBL}_s[\tau] := \mu||\mathsf{ID}(f)$ and $\mathsf{BF}_s \leftarrow \mathsf{BFAdd}(\mathsf{BF}_s, \tau)$
  end
  Store $C$ locally and set $\mathsf{state}'_s = (\mathsf{TBL}_s, \mathsf{BF}_s, \sigma, T)$
  ```

Figure 5.5: The full-fledged DSSE construction achieving forward privacy, search capability enforcement and delegated verifiability. Note that the downloaded encrypted files can be decrypted with $K_{\mathsf{SE}}$.

**Suppose the data owner generated $r \leftarrow \mathsf{SE.GenKey}$ and securely shared $r$ with authorized users and the server.**

- $\underline{\mathsf{token} \leftarrow \mathsf{GenToken}(\mathsf{K}, \mathsf{BF}_c, r, w)}$: The data owner generates the search token as follows: (i) Retrieve $(\mathsf{cnt}, \gamma_{\mathsf{cnt}})$ from $\mathsf{TBL}_c$ with respect to $w$; (ii) Compute $K_{\mathsf{cnt}} = \mathscr{F}_1(K, \mathscr{H}(w||\mathsf{cnt}))$; and (iii) Let $\mathsf{token} = \mathsf{SE.Enc}(r, \mathscr{F}_1(K, w||\mathsf{cnt})||K_{\mathsf{cnt}})$, which will be sent to the server.

- $\underline{(\mathsf{rst}, \mathsf{prof}) \leftarrow \mathsf{Search}(\mathsf{state}_s, r, \mathsf{token})}$: The server runs $\mathsf{SE.Dec}(r, \mathsf{token})$ to get $(\mathscr{F}_1(K, w||\mathsf{cnt})||K_{\mathsf{cnt}}$, and conducts the search by retrieving $\mu_{\mathsf{cnt}}||\mathsf{ID}(f)$ from $\mathsf{TBL}_s$ with respect to $\tau' = \mathscr{F}_1(K, w||\mathsf{cnt})$, computing $\mu_{\mathsf{cnt}} \bigoplus \mathscr{F}_3(K_{\mathsf{cnt}}, \tau')$ to get $\gamma_{\mathsf{cnt}}$, letting $K' = K_{\mathsf{cnt}}$, $\mathsf{prof} = (\sigma, T, \mathsf{BF}_s, \gamma_{\mathsf{cnt}})$, $\mathsf{rst} = \emptyset$, and

```
**while** $K' \neq 0^\lambda$ **do**
    Retrieve $\mu||\mathsf{ID}(f)$ from $\mathsf{TBL}_s$ with respect to $\tau'$, and let $\mathsf{rst} = \mathsf{rst} \bigcup \{\mathsf{ID}(f)\}$
    Let $\tau'||K'||\gamma' = \mu \bigoplus \mathscr{F}_3(K', \tau')$ (which results in $\mathscr{F}_1(K, w||(i-1))||K_{i-1}||\gamma_{i-1}$ if the current counter is $i$)
**end**
Return rst as the search result and prof as the proof
```

- $\underline{\mathsf{SSEVerify}(K, w, \mathsf{cnt}, \mathsf{rst}, \mathsf{prof})}$: Given $\mathsf{prof} = (\sigma, T, \mathsf{BF}_s, \gamma_{\mathsf{cnt}})$, the data owner check whether the size of rst is equal to the counter cnt or not. If not, then return 0 and abort. Otherwise, the verification proceeds as follows:
  - Given $\mathsf{ID}(f_i) \in \mathsf{rst}, 1 \leq i \leq \mathsf{cnt}$, fetch encrypted data files $C_1, \ldots, C_{\mathsf{cnt}}$ from the server.
  - If both equations hold, then output 1; otherwise output 0 (The data owner might not check Eq.(2) because of knowing correct cnt):

$$\bigoplus_{i=1}^{\mathsf{cnt}} \mathsf{Mac.GenMac}(K_{\mathsf{Mac}}, C_i||w) \overset{?}{=} \gamma_{\mathsf{cnt}} \ (1) \qquad \mathsf{Mac.GenMac}(K_{\mathsf{Mac}}, \mathsf{BF}_s||T) \overset{?}{=} \sigma \ (2)$$

Figure 5.5 (Continued): The full-fledged DSSE construction achieving forward privacy, search capability enforcement and delegated verifiability. Note that the downloaded encrypted files can be decrypted with $K_{\mathsf{SE}}$.

the server operates correctly, $\mathsf{BF}_c = \mathsf{BF}_s$ holds. Thus, only the MAC is uploaded to the server, which is then used by authorized users to check the integrity of the received $\mathsf{BF}_s$ to assure the correctness of the guessing counter value.

However, only assuring correct number of file identifiers is not enough, authorized users need to verify the correctness of the retrieved files with respect to the keyword $w$. To achieve this, each keyword is associated to an aggregate MAC, which is the result of aggregating MACs of all outsourced encrypted files containing $w$.

### 5.5.2 Main Construction

Based on the above ideas, we present the full-fledged DSSE construction as shown in Fig. 5.5, which highlights the difference from the basic construction in red color. The data owner maintains the state information (i.e., tuples of $(w, \mathsf{cnt}, \gamma_{\mathsf{cnt}})$) with a hash table $\mathsf{TBL}_c$ mapping $w$ to $\mathsf{cnt}, \gamma_{\mathsf{cnt}}$, where $\gamma_{\mathsf{cnt}}$ is the aggregation of the MAC for the concatenation of the file and $w$ so far. The reason of concatenating the file and $w$ as input, rather using the file itself, is to prevent the replacement attack: given keyword $w_1$, the server might intentionally return the search result for another keyword $w_2$, an aggregate MAC and the set of file identifiers, which has the same number of file identifiers as that for keyword $w_1$.

Also, the data owner uses the timestamp $T$ (together with the Bloom filter $\mathsf{BF}_c$) to generate the MAC for preventing the replaying attack that the server might possibly return stale search result. We implicitly leverage the fact that the new file is periodically uploaded (e.g., every 10 minutes), so that authorized users can use the timestamp $T$ to assure the aggregate MAC is newly generated by the data owner.

Due to the lack of knowledge about cnt, authorized users (other than the data owner) generate the search token as shown in Fig. 5.6, where the WHILE loop is to guess the counter value. Note that with the guessing counter value and the shared key from the data owner, authorized users are able to run SSEVerify to verify the correctness of the research result.

**Optimization II: Speed up guessing the latest counter with binary search.** Instead of guessing

```
token ← GenToken(K, BF_s, r, w): After fetching the Bloom filter BF_s from the server, the authorized data user generates the search token as follows:
    Let cnt = 1;
    while TRUE do
        $\tau_{\mathsf{cnt}} = \mathscr{F}_1(K, w||\mathsf{cnt})$
        if $\mathsf{BFVerify}(\mathsf{BF}_s, \tau_{\mathsf{cnt}})$ outputs 1 then
            $\mathsf{cnt} = \mathsf{cnt} + 1$
        else
            $\mathsf{cnt} = \mathsf{cnt} - 1$
            break;
        end
    end
    Compute $K_{\mathsf{cnt}} = \mathscr{F}_1(K, \mathscr{H}(w||\mathsf{cnt}))$;
    Let $\mathsf{token} = \mathsf{SE.Enc}(r, \mathscr{F}_1(K, w||\mathsf{cnt})||K_{\mathsf{cnt}})$, which will be sent to the server.
```

Figure 5.6: The algorithm for the authorized user generating search token. The data owner has already distributed $K = (K, K_{\mathsf{SE}}, K_{\mathsf{Mac}})$ and $r$ to the authorized user.

the counter value linearly, authorized users can use the binary search to accelerate the guessing: The authorized user sets a large enough upper bound $Max$, and conducts the binary search for the latest counter $\mathsf{cnt}$ within $[1, Max]$ such that $\mathscr{F}_1(K, w||\mathsf{cnt})$ is an element hashed to $\mathsf{BF}_s$ while $\mathscr{F}_1(K, w||\mathsf{cnt}+1)$ not.

**Optimization III: Reduce the number of elements hashed to $\mathsf{BF}_s$.** Note that the number of elements hashed into $\mathsf{BF}_s$ might become huge due to the increasing counter value $\mathsf{cnt}$ when generating $\mathscr{F}_1(K, w||\mathsf{cnt})$ for keyword $w$. This results into a drawback: In order to keep low false positive rate, the size of $\mathsf{BF}_s$ becomes very large, which incurs costly bandwidth when authorized users retrieve it from the server. To get rid of it, the "regular update" strategy can be used:

- Given the state information $\mathsf{TBL}_c$, the data owner regularly (e.g., annually) generates a new Bloom filter $\mathsf{BF}_c$, which implicitly stores the current counter $\mathsf{cnt}_L$ for each keyword $w$, generates the MAC and sends $\mathsf{BF}_c$ and the MAC to the server.

- The server lets $\mathsf{BF}_s = \mathsf{BF}_c$ and proceeds as in Fig. 5.5.

- After receiving $\mathsf{BF}_s$, the authorized user extracts the counter $\mathsf{cnt}_L$ first, and then guesses the latest counter starting from $\mathsf{cnt}_L$.

By doing this, $\mathsf{BF}_s$ only contains elements with counters beginning with $\mathsf{cnt}_L$ (rather than from 1) for keyword $w$, and therefore its size can be reduced when keeping the same false positive rate. In addition, implicitly storing $\mathsf{cnt}_L$ for keyword $w$ in $\mathsf{BF}_c$ can be done as follows: Given $\mathsf{cnt}_L$, the data owner hashes $\mathscr{F}_1(K, w||\mathsf{pos}||\mathsf{digit}_{\mathsf{pos}})$ to $\mathsf{BF}_c$ where $\mathsf{digit}_{\mathsf{pos}}$ is the least significant digit of $\mathsf{cnt}_L$ when $\mathsf{pos} = 1$, and the authorized user can guess $\mathsf{cnt}_L$ by enumerating the combination of $\mathsf{pos} = 1, \ldots$ and $\mathsf{digit}_{\mathsf{pos}} = 0, \ldots, 9$. For example, given $\mathsf{cnt}_L = 456$ for keyword $w$, $\mathscr{F}_1(K, w||1||6), \mathscr{F}_1(K, w||2||5)$ and $\mathscr{F}_1(K, w||3||4)$ were hashed to $\mathsf{BF}_c$. Authorized users can guess $\mathsf{cnt}_L$ by enumerating $\mathsf{pos}$ and $\mathsf{digit}_{\mathsf{pos}}$ and checking $\mathsf{BF}_s$ (since $\mathsf{BF}_s = \mathsf{BF}_c$) to determine whether $\mathscr{F}_1(K, w||\mathsf{pos}||\mathsf{digit}_{\mathsf{pos}})$ has been hashed into $\mathsf{BF}_s$ or not, until that there exists some $\mathsf{pos}$ such that none of elements $\mathscr{F}_1(K, w||\mathsf{pos}||\mathsf{digit}_{\mathsf{pos}}), \mathsf{digit}_{\mathsf{pos}} = 0, \ldots, 9$, was hashed into $\mathsf{BF}_s$.

## 5.6 Security Analysis

We evaluate the security of our full-fledged construction to show that it achieves the security goals described in Section 5.2.3.

**Data confidentiality**. The outsourced files are encrypted with the secure symmetric encryption together with secret key $K_{\mathsf{SE}}$. Without leaking $K_{\mathsf{SE}}$ to the server, data confidentiality is naturally assured by the secure symmetric encryption.

**Index confidentiality**. Since each keyword in the index (i.e., $state_s$) is encrypted by the secure pseudorandom function $\mathscr{F}_1$, without knowing the secret key, the server cannot learn the keyword from the index.

**Forward privacy**. As discussed in the Section 5.5, our construction encrypts the combination of the increasing counter and the keyword together, which makes the server unable to link the keyword in the newly added file to any stored encrypted keyword, without knowing the secret key $K$. In addition, a secure pseudorandom function is used to mask the connection of tuples generated from the same keyword but with consecutive counter values, without knowing the corresponding secret key, the server cannot correlate these tuples together. That is, the server cannot know whether the newly added file contains any stored encrypted keyword, without knowing the secret keys for the pseudorandom functions.

**Search token privacy**. The keyword associated with the search token is protected with a secure pseudorandom function. Without knowing the key $K$, the server cannot learn the keyword.

**Search capability enforcement**. Our construction implicitly shares the state information using the Bloom filter, and uses the group key to assure that only authorized users can generate valid search tokens. Therefore, the data owner can enforce the search capability securely (note that the cloud and users are not allowed to collude in our assumption).

**Verifiability**. Our construction uses the timestamp and the MAC to assure the freshness and correctness of Bloom filter $\mathsf{BF}_s$, which further assures the correctness of the counter value for any keyword, forcing the server honestly returning correct number of the encrypted files. Moreover, the construction uses the aggregate MAC to assure the integrity of the returned files with respect to the keyword. Therefore, given the secure message authentication scheme, our construction assures that the authorized users and data owner can correctly verify the returned search result with an overwhelming probability.

## 5.7 Performance Evaluation

In this section, we present the empirical performance result by simulating the e-healthcare system with the full-fledged DSSE implementation.

**Implementation:** We implemented the full-fledged DSSE in JAVA, and instantiated $\mathscr{F}_1, \mathsf{Mac}$ with HMAC-SHA-1, $\mathscr{F}_3$ with HMAC-SHA-512, SE with AES and $\mathscr{H}$ with SHA-1. In addition, we implemented all optimizations as mentioned above. We simulated the e-healthcare system by developing three separate processes for the data owner, the server and the authorized user respectively. The three processes communicate with each other via RESTful API, and were running in a laptop with 2.5GHz Intel i5 CPU, 8GB RAM and MAC OS.

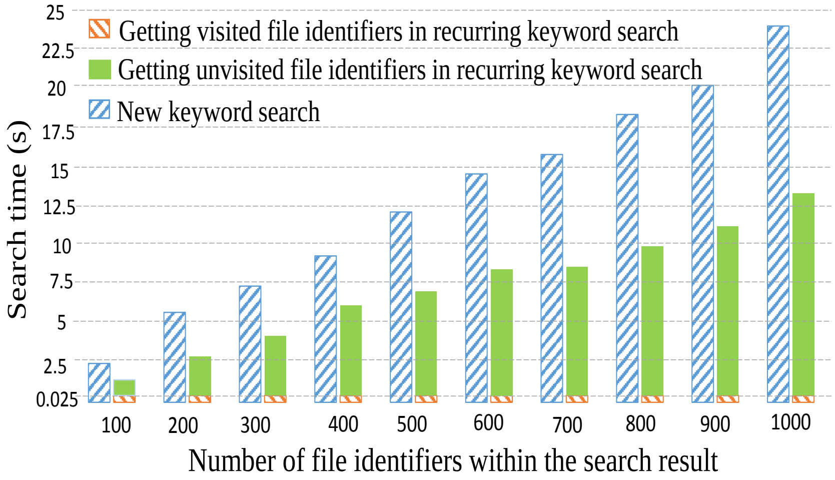


Figure 5.7: Performance for search operation running by the server storing one million files and the corresponding index (i.e., $\mathsf{state}_s$). Note that for recurring keyword search, half the number of file identifiers in the search result were newly added since the last time of the same keyword query (called unvisited identifiers), and the other half has been added to the server before the last time of the same keyword query (called visited identifiers).

**Dataset:** In the experiment each PHI file consists of 15 pairs of attribute[1] in the format of attribute:value, which is treated as one single keyword (e.g., $w = heartbeat : 75$). To simulate the scenario that the IoT gateway assembles and uploads a new PHI file in every 10 minutes and lasts for 20-year, 1,051,200 synthesized PHI files were uploaded.

**Performance on the Data Owner.** The average time for the data owner running AddFile is 190 milliseconds, and the size of hash table (i.e, $\mathsf{TBL}_c$) is around 1.3MB after uploading one million PHI files. The data owner also maintains a Bloom filter (i.e., $\mathsf{BF}_c$) of around 5MB by setting the false positive rate as $2^{-30}$, and updates it every year (i.e., after adding $144 \times 365 = 52,560$ new files as in Optimization III). We note that if the Bloom filter can be updated more frequently (e.g., less than every year), the size of the Bloom filter can be further reduced.

**Performance on the Server.** We note that given a search token for keyword $w$ at time $T_{i+1}$, the complexity of running Search is linear to the number of encrypted files that contain $w$ and were uploaded within the interval of $T_i$ and $T_{i+1}$ (sublinear to the number of encrypted files), where $T_i$ is

[1] The attributes include heartbeat, blood sugar, blood pressure, temperature and so on as in `http://www.clouddx.com/downloads/Heart-Friendly-Report-2015-12-24-092313.pdf`

the last time when $w$ was searched for (The time of initializing the system can be regarded as $T_0$, at which the search result for any keyword is null). The reason is that, with Optimization I, the server stores in a consecutive manner all identifiers of encrypted files having $w$ at $T_i$, and can access them in a constant time thereafter. Therefore, we evaluated the search performance in the two scenarios: (i) new keyword search, which simulates that keyword $w$ has never been queried before, and (ii) recurring keyword search, which simulates that keyword $w$ has been queried before. Fig. 5.7 shows the performance. We can see that the search performance is closely related to the number of newly added files within the interval between two consecutive queries for the same keyword. In addition, we can see that the search performance is quite practical since returning 100 files identifiers for new keyword search (resp. recurring keyword search) only costs around 2 seconds (resp. 1 second) (note that one million files and the corresponding index were stored in the server).

**Performance on the Authorized User.** The time for the authorized user generating search token can be neglected (approximately 10 ms) due to the binary search (Optimization II). Therefore, we concentrated on the execution time for the authorized user verifying the correctness of the search result. The performance result is shown in Fig. 5.8, where we divided the verification time into two parts: one is for verifying the correctness of the Bloom filter (i.e., $\mathsf{BF}_s$ retrieved from the server) and the other one is for verifying the aggregate MAC over all returned files. We can see that the time for verifying the correctness of the Bloom filter is quite similar (e.g., around 55 ms in our experiments) no matter how many files are within the search result, and the time of verifying the aggregate MAC over all returned files is linear to the number of files. We can see that verification is practical because, even when dealing with the search result having 1,000 files, the verification time is only around 135 ms.

## 5.8 Related Work

Cloud-assisted IoT system has become a popular design paradigm in many applications [138, 104, 140, 157], since the powerful computation and storage capabilities of cloud can overcome the

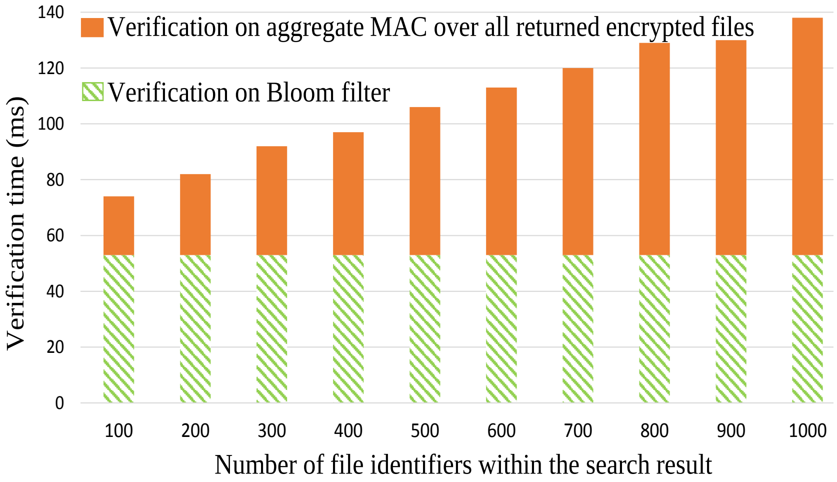


Figure 5.8: Performance for the authorized user verifying the correctness of the search result, i.e., verifying the correctness of the Bloom filter and aggregate MAC over all returned files.

constrains of IoT devices. This paper particularly relates to searchable encryption in e-healthcare:

**Searchable Encryption**. Song *et al.* [134] first explored the problem of searchable symmetric encryption and presented a scheme with linear search time. Curtmola *et al.* [56] gave the first inverted index based scheme to achieve sub-linear search time. Although this scheme greatly boosts search efficiency, it does not support dynamic dataset. Since then, several schemes [94, 93, 118, 47, 81, 135] about dynamic SSE have been proposed, among which [94, 93, 118, 47] fail to provide forward privacy. Moreover, the previous work in [135] offers forward privacy using a complicated hierarchical data structure, whereas the contribution in [81] only achieves limited forward privacy (i.e., leaks the keywords contained in a new file if they have been searched for in the past). Besides dynamic SSE, verifiable SSE have been studied by [98, 165, 52, 137], which enables users to verify search results by using some verifiable structure such as the Merkle tree or an accumulator. However, previous work did not pay special attention to dataset with sequentially added files, which might leak additional information during the process of updating verifiable structure.

**Secure data storage for e-healthcare**. Several searchable encryption schemes [138, 104, 140] have been proposed for e-healthcare applications. Tan *et al.* [138] proposed a lightweight IBE scheme to encrypt the sensing data and store it on a cloud. However, their public-key based scheme makes search over encrypted data very inefficient. Li *et al.* [104] presented an authorized search scheme over encrypted health data, which aims to realize search in a multi-user setting by enforcing fine-grained authorization before performing search operations. However, their search scheme is based on the predicate encryption, which is less efficient than SSE. Tong *et al.* [140] proposed a SSE-based healthcare system, which achieves high search efficiency and partially hides the search and access patterns by using the redundancy. However, their scheme depends on a trusted private cloud and is not able to support dynamic data.

## 5.9 Summary

In this chapter, we proposed a reliable, searchable and privacy-preserving e-healthcare system. The core of our system is a novel and full-fledged dynamic SSE scheme with forward privacy and delegated verifiability, which is dedicatedly designed to protect sensitive PHI files on cloud storage and enable HSPs to search on the encrypted PHI under the control of patients. The salient features such as forward privacy and delegated verifiability are achieved by a unique combination of the increasing counter, Bloom filter and aggregate MAC. Our experimental results and security analysis demonstrate that the proposed system provides a promising solution for meeting the stringent security and performance requirements of the healthcare industry in practice.

# Chapter 6

# Cloud-Assisted Machine Learning Classification over Encrypted Data for IoT Devices

## 6.1 Introduction

In the previous chapter, we propose a symmetric searchable encryption scheme with forward privacy for IoT data, which enables IoT clients to store their data on the cloud in an encrypted format, and meanwhile allows third parties, such as primary doctor, to retrieve the data and analyze the data. However, it introduces another drawback. That is, to enable third parties to analyze the data, we have to show them the cleartexts, but that is what we do not know if we cannot fully trust them. Nowadays, not only doctors with expertises can provide healthcare services, machine learning companies also want to offer diagnostic or predictive services. For example, DeepMind is planing to and make its healthcare AI charge by results [6]. However, clients have big concerns to give their very private data to a company to realize data analysis. Therefore, in this chapter, to tackle this privacy concern, we explore how to analyze the data without seeing the cleartexts for IoT data.

Recall that IoT integrates the ubiquitous sensing capability with advanced computing and data analysis capabilities of backend applications to provide automated extraction of insights by making sense of plethora of data generated by these sensors. This has led to a pervasive deployment of intelligence into our daily life, ranging from healthcare (e.g., remote patient monitoring, wearable fitness tracking) and security (e.g., community monitoring) to home automation, smart communities and smart cities (e.g., smart traffic control, distributed pollution monitoring).

In the IoT platform, the frontend IoT devices are usually resource-constrained, which have very limited storage and computing power to support complicated computations. Therefore, the intelligence is often provided by machine-learning applications running on powerful backend servers. With the large volume of data generated by IoT devices, backend servers train predictive models to map feature vectors to categorical or real-valued outputs. Taking future sensing data as the input, the models output classification or predication results to the clients. For example, wearable devices with accelerometer and gyroscope, depth cameras, etc. are deployed in the home of old adults for fall detection [136, 74]. With the realtime sensor data, the predictive model trained at cloud servers analyze the vertical state of objects to detect a fall and notify caregivers almost in a ubiquitous way.

The success of machine learning (ML) on IoT platforms has led to an explosion of demands. In fact, providing classification and predication services that are customized to different application domains is becoming an emerging business paradigm, known as "*machine-learning-as-a-service*" (MLaaS). For example, many companies provide automatic medical assessments and risk profiling service by analyzing the physical measurements collected by the wearable fitness tracking device of a client. To simplify and broaden machine learning model deployment, major cloud service providers, such as Amazon, Google, Microsoft, BigML, *etc.* are offering cloud-based MLaaS, which trains predictive models and charges future usage of the model at a pay-per-query or subscription-based cost.

However, IoT applications atop MLaaS raise several privacy concerns, which may hinder further adoption of this emerging business paradigm. Privacy involved in the described application scenario is two-fold: privacy of the subject (i.e., data owner) and privacy of the ML model. In many applications, data collected by IoT devices such as electrocardiogram or environmental temperature are sensitive or proprietary data of its owner. ML models trained on the sensitive or proprietary data are also sensitive or proprietary. Revealing models may leak information about the underlying training data [32, 68]. Additionally, predictive models (e.g., type, structure and parameters) are core intellectual property items of machine learning service providers (MLSPs). However, without proper protection, these models are under the risk of model extraction attacks, which query a prediction API to reverse-engineer model characteristics and extract an equivalent or near-equivalent model of the original one [142]. Therefore, the ML models need to be well protected from being leaked. Meanwhile, for privacy protection purposes, the client may not want to reveal her sensitive data measured by an IoT device to MLSP, during the use of predictive models. The concern comes from clients' trust to MLSP regarding not only their appropriate use of the data, but also their protection to the data considering the numerous data breaches reported in recent years [97].

Therefore, it is important to keep both the data and the ML models private in IoT applications that handle sensitive or proprietary data. Ideally, this requires the ML model to take data from the IoT device of the client in the encrypted form as input, and generate predication or classification results as the output. The output is still in the encrypted form that only the client can recover. Furthermore, in the end of the classification, the client only knows the classification result but nothing else about the model, and MLSP learns nothing about the client's input nor the classification result. The problem has recently been studied in the literature of *privacy-preserving classification* [44, 43, 154].

Supervised classification algorithms typically consists of two phases: the training phase, in which the algorithm learns a model $w$ from the dataset, and the classification phase, in which the input data, known as feature vector $x$, is fed into the trained classifier $C$, which returns the classification result by applying the model as $C(x,w)$. Privacy-preserving classification assumes the model (i.e., the classifier $C$) has already been trained through privacy-preserving training approaches (please refer to Section 7.6 for details), and thus focuses on the second phase, in which MLSP provides privacy-preserving classification service to the clients. Existing solutions often adopt the classic client-server setting to establish a communication protocol between the two parties and conduct several rounds of secure multi-party computation to realize privacy-preserving classification. However, these approaches either lack the protection to the models [43], or introduce a large communication and computation overhead at the client side [44, 154], which make them impractical for IoT applications.

To provide privacy-preserving classification service on IoT platforms, we need to reduce the computation and communication overhead at the resource-constrained IoT device. To tackle this problem, we propose a novel *cloud-assisted privacy-preserving classification* approach, which introduces an additional cloud as a third party in the model and outsource a part of computationally expensive operations from the client (i.e., IoT devices). Moreover, we develop the communication protocol among the three parties, which avoids unnecessary communications between the client and the other two and thus reduces the participation of the client in the protocol.

## 6.2 Problem Formulation

### 6.2.1 System Model

In this work, we focus on a popular classification algorithm, called *hyperplane decision-based classifiers*, which is a parametric and discriminative classifier widely used in practice through various instances, such as support vector machines (SVM), logistic regression and least squares. Generally speaking, the hyperplane decision-based classifier consists of $k$ weight vectors, corresponding to $k$ distinct classes: $\hat{\mathbf{W}} = \{\mathbf{W}_1, \ldots, \mathbf{W}_k\}$, where $\mathbf{W}_i \in \mathbb{R}^d$ is a vector of $d$ dimensions.

To determine which class the input $\mathbf{Y}$ belongs to, the classifier evaluates the index $k^*$ such that

$$k^* = \underset{1 \le i \le k}{\operatorname{argmax}} \langle \mathbf{W}_i, \mathbf{Y} \rangle$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product operation.

The proposed three-party privacy-preserving classification scheme for IoT applications is shown

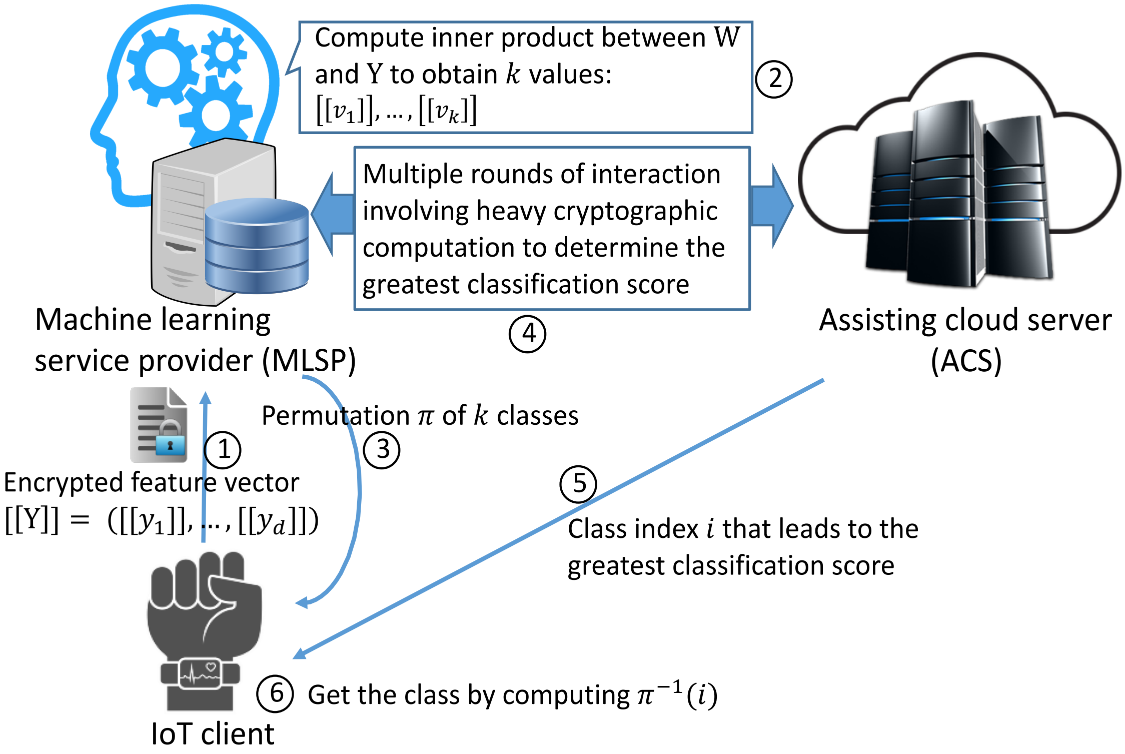

Compute inner product between W and Y to obtain $k$ values: $[[v_1]], \ldots, [[v_k]]$
②
Multiple rounds of interaction involving heavy cryptographic computation to determine the greatest classification score
④
Machine learning service provider (MLSP)
Assisting cloud server (ACS)
Permutation $\pi$ of $k$ classes
③
①
Encrypted feature vector
$[[Y]] = ([[y_1]], \ldots, [[y_d]])$
⑤
Class index $i$ that leads to the greatest classification score
⑥ Get the class by computing $\pi^{-1}(i)$
IoT client

Figure 6.1: The system model of three-party privacy-preserving classification system.

in Fig. 6.1. The system consists of three entities: an IoT device (i.e., the client), a machine learning service provider (MLSP), and an assisting cloud server (AS). The IoT device, e.g., a smart watch or a smart gateway, is generating sensitive data about the client, from which a feature vector of $d$ dimensions is extracted as $\mathbf{Y} = (y_1, \cdots, y_d) \in \mathbb{R}^d$. Since the extracted feature vector is sensitive, the device encrypts it so that it is protected from any other entity in the system. $[\![\cdot]\!]$ in step 1 denotes each element of the vector is in the encrypted form. MLSP holds the machine learning model to provide the classification service by taking $\mathbf{Y}$ as input. The AS is included so that heavy communication/computation overhead can be delegated from the IoT devices to AS.

### 6.2.2 Threat Model

As assumed in most previous work on privacy-preserving "machine-learning-as-a-service" work [44, 43, 154], the user (i.e., the IoT device), MLSP and AS are generally "honest-but-curious", meaning that they can faithfully perform the protocol while trying their best to infer any private information from other parties. To be more specific, because the machine learning model is regarded as the core intelligence property, the user is curious about the machine learning model held by MLSP. On the other hand, MLSP tries to sniff the user's input and the classification result. Note that AS is interested in the private input from the user and the machine learning model from MLSP. Finally, we assume that there is no collusion between MLSP and AS. Note that either MLSP or AS may output incorrect results to the user due to accidental errors or malicious attacks. While this problem could be solved by employing existing verifiable computation techniques, it is out of the scope of this paper.

### 6.2.3 Design Goals

In this work, we aim to design a light-weight and mutually privacy-preserving approach in the paradigm of "machine-learning-as-a-service" for IoT devices. In terms of security, our approach should minimize the privacy leak without requiring mutual trust between user and MLSP. In addition, due to the limited computation and storage capacity of IoT devices owned by the user, it is expected that the overhead on the user side should be as small as possible. Therefore, our design goals include two aspects:

1. **Security**. After the classification is finished, the user should only know the classification result but nothing else about the machine learning model. On the other hand, MLSP should not know any information regarding the model's input and output. Moreover, AS should know nothing at all.

2. **Lightweight operations on IoT device**. The operations on the user should be minimized so that resource-constrained IoT device could afford it.

## 6.3 Preliminaries

### 6.3.1 Two-stage decryption Paillier-based cryptosystem

Paillier cryptosystem is a probabilistic asymmetric algorithm proposed by Pascal Paillier [122] for public key cryptography. A notable feature of the Paillier cryptosystem is the homomorphic properties along with its non-deterministic encryption. More specifically, the encryption function is additively homomorphic, so the the product of two ciphertexts will decrypt to the sum of their corresponding plaintexts, namely, $Dec(Enc(m_1) \cdot Enc(m_2)) = m_1 + m_2$.

Ateniese et al. [31] proposed a variant of Paillier cryptosystem built from two-trapdoor Paillier [55, 45] with the promising properties: (i) it inherits the additively homomorphic property, and (ii) the ciphertext decryption can be done in a two-stage procedure. The second property is essentially useful in our context because it allows the private key owner to delegate the decryption capability to a third party in a control manner. In this way, the private key owner can control one stage decryption so that without tight cooperation, any third party only holding one stage decryption key cannot fully decrypt the ciphertext. To be specific, the model for the cryptosystem involves three parties: sender, receiver and a proxy, where the sender encrypts the message with the receiver's public key. The proxy first partially decrypts the ciphertext and then the receiver finally recovers the plaintext from the transformed ciphertext. The proposed scheme consists of the following five functions $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{DirectDec}, \mathsf{Transform}, \mathsf{FinalDec})$:

- KeyGen: This algorithm generates the public/private keys for the receiver, where $x$ is the private key, $g^x$ is the public key, $x_1$ and $x_2$ are the private shares of $x$ such that $x = x_1 + x_2$.

- Enc: This algorithm encrypts the plaintext with the input of the receiver's public key $g^x$.

- Dec: This algorithm decrypts the ciphertext with the private key $x$.

- Transform. This algorithm partially decrypts the ciphertext with the private key share $x_1$.

- FinalDec: This algorithm decrypts the transformed ciphertext to the plaintext with the private key share $x_2$.

### 6.3.2 Privacy-preserving integer comparison protocol

Another building block of our model is the private-preserving integer comparison protocol, where two parties, each possessing a respective integer, jointly compare their integers while without revealing one's own integer to each other. More specifically, we adopt the DGK comparison protocol proposed by Damgård, Geisler and Krøigaard [57]. Suppose that party A holds $x$ and party B holds $y$ where $x$ and $y$ are two integers of $l$ bits. The high-level idea of the protocol is that: The integer can be denoted by the binary representation such that $x = x_1x_2\cdots x_l$ and $y = y_1y_2\cdots y_l$ respectively, and $x < y$ holds if and only if there exist an index $i \in [1,l]$ such that $x_i < y_i$ and for all $j < i$, $x_j = y_j$. In another word, if there exist $i \in [1,l]$, such that $z_i = x_i - y_i + 1 + \sum_{j<i}(x_j \oplus y_j) = 0$, then $x < y$. Based on this idea, the protocol runs as follows (informal): A encrypts each bit of $x$ using an additively homomorphic encryption scheme with A's public key and sends the bitwise encryptions to B. B homomorphically computes encryptions of $z_1r_1,\dots,z_lr_l$ where $r_i, 1 \le i \le l$ are randomly chosen from $\mathbb{Z}_n$. Note that the encryption of $z_i$ can be computed over the ciphertexts sent from A, because $x_i \oplus 0 = x_i$ and $x_i \oplus 1 = 1 - x_i$ where $y_i$ is known to B. B sends the ciphertexts back to A in a random order, so that A can decrypt all ciphertexts to see whether there exists a single ciphertext decrypted to 0 and $l-1$ ciphertexts decrypted to non-zero random numbers.

## 6.4 Three-party Privacy-preserving classification

In this section, we present our proposed solution which is to securely evaluate the hyperplane-based classifier:

$$k^* = \underset{1\le i\le [k]}{\operatorname{argmax}} \langle \mathbf{W}_i, \mathbf{Y} \rangle.$$

This classifier typically supports binary classification ($k = 2$): given a user's input $Y$, we label $Y$ as an instance of class $c_1$ if $\langle W, Y \rangle \ge 0$, otherwise, we label it to class $c_2$. To extend the binary

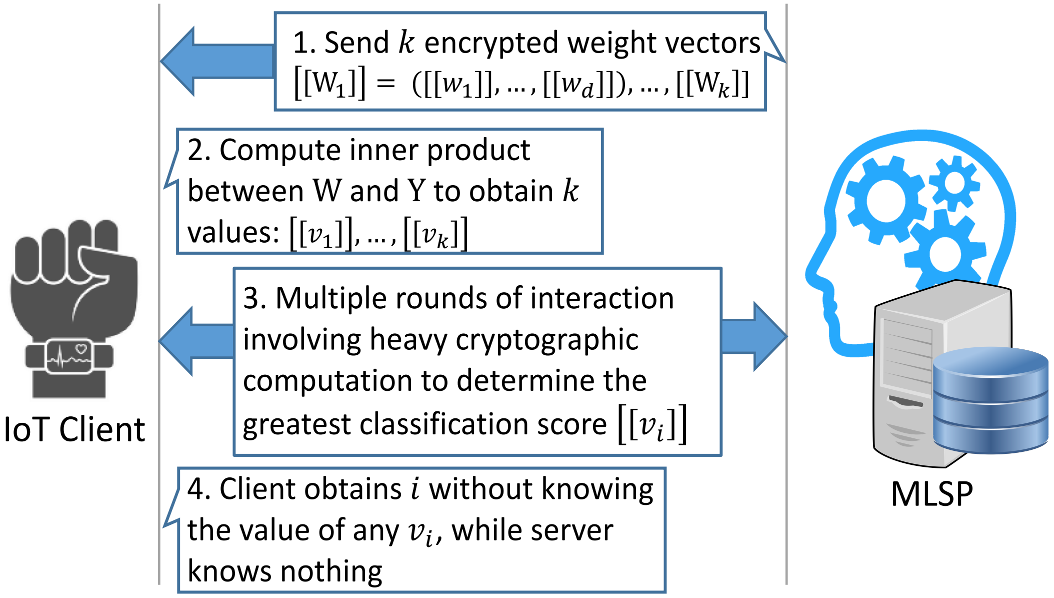

1. Send $k$ encrypted weight vectors
$[[W_1]] = ([[w_1]], \ldots, [[w_d]]), \ldots, [[W_k]]$
2. Compute inner product between W and Y to obtain $k$ values: $[[v_1]], \ldots, [[v_k]]$
3. Multiple rounds of interaction involving heavy cryptographic computation to determine the greatest classification score $[[v_i]]$
4. Client obtains $i$ without knowing the value of any $v_i$, while server knows nothing
IoT Client
MLSP

Figure 6.2: Two-party model proposed by Bost et al. [44].

classifier to the one supporting $k$ classes, one of the most common methods is adopt, $one-versus-all$, namely, $k$ different binary classifiers are trained, each of which is used to distinguish one from others. So, the index $i$ that leads to the greatest classification score can finally determine the classification result out of $k$ classes.

Therefore, in order to protect the privacy for the user's data and MLSP' classification model respectively, evaluating this classifier demands two basic protocols : (i) the privacy-preserving inner product protocol that can securely compute the inner product of $\mathbf{W}_i$ and $\mathbf{Y}$ without leaking the vectors to each other, and (ii) the *argmax* protocol which is to privately determine which index $i, 1 \leq i \leq [k]$, leads to the greatest inner product.

### 6.4.1 Design Rationale

Before talking about the details of our three-party scheme, let us first see how this problem is addressed in the current two-party setting, i.e., client and MLSP. Fig. 6.2 shows the two-party scheme proposed by Bost et al [44]: MLSP holding the key pair sends $k$ encrypted weight vectors to the client in Step 1. Then, client homomorphically computes $k$ inner products between weight

**User Input** : $\mathbf{Y} = (y_1, \cdots, y_d) \in \mathbb{Z}^d$, secret key $SK_A$ and public key $PK_A$
MLSP **Input** : $\mathbf{W} = (w_1, \cdots, w_d) \in \mathbb{Z}^d$, public key $PK_A$
MLSP **Output**: $[\![\langle \mathbf{W}, \mathbf{Y} \rangle]\!]$

1. The user encrypts $y_1, \cdots, y_d$ and sends the encryptions $[\![y_1]\!], \cdots, [\![y_d]\!]$ to MLSP.
2. MLSP performs the private inner product by computing $[\![v]\!] = \prod_i [\![y_i]\!]^{w_i} \bmod N^2$.
   // $v = \sum w_i y_i$
3. MLSP outputs $[\![v]\!]$, which is used as the input to the *argmax* protocol.

Figure 6.3: Privacy-preserving inner product

vector and his input vector. After obtaining $k$ values, client wonders which one is the greatest but he should not know the exact value. Since the client does not have the private key, he needs the help from MLSP to finish this task. To avoid revealing the relative order and thus the classification result to MLSP, the client cannot send $k$ values back. To achieve this goal, the client needs to blind the values and compare every pairs with the help of MLSP, which is a multiple rounds of interaction involving heavy cryptographic computation. Interested readers please refer to [44].

However, the huge cost of computation and communication imposed to the client device in Step 3 of Fig. 6.2 may overwhelm its capacity, especially resource-constrained IoT devices. Other two-party schemes [154] also require the client to participant in the laborious process except [43] adopting fully homomorphic encryption, which is considered several magnitudes slower than partially homomorphic schemes used in ours and [44, 154]. To the best of our knowledge, no existing schemes propose a light-weight solution for IoT devices. Therefore, in our scheme, we introduce the assisting cloud server (AS) to mitigate the overhead on the IoT devices and meanwhile achieving the security goals. To implement our scheme, the privacy-preserving integer comparison protocol mentioned in Section 6.3.2 is used as a sub-building block in Step 4 of Fig. 6.1. Besides, by leveraging the nice property of two-stage Paillier mentioned in Section 6.3.1, we outsource the costly computation to AS.

### 6.4.2 Compute Inner Product over Ciphertext

When running the classification service, the user owns a sensitive feature vector (which is extracted from some raw data) and the server holds the private machine learning model. In order to achieve the security goals mentioned in Section 6.2.3, assuming there exists an additively homomorphic encryption scheme (we can instantiate it with Paillier-based cryptosystem as in Section 6.3.1), the inner product could be computed with two different alternatives: (1) User-initiated: The user generates a pair of public/private key, encrypts the feature vector with the public key and uploads the encrypted input to MLSP, which in turn performs the private inner product; (2) MLSP-initiated: Inversely, MLSP generates the public/private keys, encrypts the machine learning model, which is a set of weight vectors, and sends those encrypted vectors to the user, such that the user is able to compute the inner product together with his own feature vector.

Bost et al. adopt the MLSP-initiated strategy (corresponding to Step 1 of Fig. 6.2), while we choose the user-initiated approach. A potential benefit of MLSP-initiated strategy is that, since the overhead of the aggregation operation used in computing inner product is more light-weight than the encryption operation, it seems to incur less computation to the user. However, let us take a closer look at their scheme: at the end of the classification, the encryptions of user's data still be stored on the user side. After a long period, the storage on the client will be exhausted, which will be worse for the resource-constrained IoT devices. To avoid exhausting storage, two options are available now: (1) Discard the historical data after getting the classification result. However, it violates one principle of IoT applications, namely, value comes from the data. Once the historical data is discarded, even if machine learning techniques are improved greatly in the future, it is impossible to further study over a long period of data, which is nevertheless very valuable particularly for healthcare applications. (2) So, a practical solution is to upload the historical data to a remote storage server such as cloud. However, to achieve strong security guarantee, all outsourced data should be encrypted before leaving the client. In this regard, it is inevitable for the user to encrypt the data, so our scheme actually does not introduce extra overhead to the IoT device. Therefore, we argue that our design is more practical for the real IoT applications.

In Fig. 6.3, we present an instantiation that implements the user-initiated strategy by using the aforementioned two-stage Pallier cryptosystem. More specifically, the client performs the KeyGen and Enc algorithms (Step 1 of Fig. 6.1):

- KeyGen: Given two large safe primes $p$ and $q$, such that $p$ and $q$ are primes of the form $p = 2p' + 1$, $q = 2q' + 1$, where $p'$ and $q'$ are also primes, the algorithm computes $n = pq$, selects $x \in [1, n^2/2]$ uniformly at random and sets the public key as $(n, g, h = g^x)$ where $g$ of order $\lambda(n) = 2p'q'$ (As remarked in [55], such that $g$ can be easily generated by selecting a random value $a \in \mathbb{Z}_{n^2}^*$ and setting $g = -a^{2n}$.), and the private key as $x$. In order to enable the two-stage decryption, the private key $x$ can be divided into two shares $x_1$ and $x_2$ such that $x = x_1 + x_2$. The client holds the private key $x$, and distributes one private key share $x_1$ to MLSP and another share $x_2$ to AS securely.

- Enc: To encrypt a message $m \in \mathbb{Z}_n$, it selects $r$ uniformly at random from $\mathbb{Z}_{n^2}$ and computes the ciphertext $C = (c_1, c_2)$ where

$$c_1 = g^r \bmod n^2, c_2 = h^r(1 + mn) \bmod n^2.$$

For the simplicity of exposition, we denote the ciphertext of value $y$ as $[\![y]\!]$, which actually consists of two parts: $c_1$ and $c_2$. As we can see, MLSP holds its model (i.e., a set of weight vectors), and performs the inner product computation together with the user's input (i.e., an encrypted feature vector), which is referred to Step 2 of Fig. 6.1. Here, we leverage the homomorphic property of the two-stage Paillier, which states that: **(1)** the product of two ciphertexts will decrypt to the sum of their corresponding plaintexts, and **(2)** an encrypted plaintext raised to a constant $k$ will decrypt to the product of the plaintext and the constant, namely, $\mathsf{Dec}(\mathsf{Enc}(m_1) * \mathsf{Enc}(m_2)) = m_1 + m_2$ and $\mathsf{Dec}(\mathsf{Enc}(m)^k) = km$. The correctness is briefly proved as follows:

$$
\begin{aligned}
&\mathsf{Enc}(m_1) = (g^{r_1} \bmod n^2, h^{r_1}(1+m_1 n) \bmod n^2)\\
&\mathsf{Enc}(m_2) = (g^{r_2} \bmod n^2, h^{r_2}(1+m_2 n) \bmod n^2)\\
&\mathsf{Enc}(m_1) * \mathsf{Enc}(m_2) = (g^{r_1+r_2} \bmod n^2, h^{r_1+r_2}(1+(m_1+m_2)n + m_1 m_2 n^2) \bmod n^2\\
&\qquad\qquad = \mathsf{Enc}(m_1+m_2) \qquad\qquad\qquad\qquad \mathbf{(1)}\\
&\mathsf{Enc}(m_1)^k = (g^{r_1 k} \bmod n^2, h^{r_1 k}(1+m_1 n)^k \bmod n^2)\\
&\qquad = (g^{r_1 k} \bmod n^2, h^{r_1 k}(1+k m_1 n + \binom{k}{2} {m_1}^2 n^2 + \cdots + {m_1}^k n^k) \bmod n^2)\\
&\qquad = (g^{r_1 k} \bmod n^2, h^{r_1 k}(1+k m_1 n) \bmod n^2) \quad = \mathsf{Enc}(k m_1) \qquad \mathbf{(2)}
\end{aligned}
$$

where the Dec algorithm is defined as:

- Dec: Given the private key $x$, the ciphtertext $C$ could be decrypted as:

$$m = L(c_2/c_1^x \bmod n^2),$$

where $L(u) = \frac{u-1}{n}$ for all $u \in \{u < n^2 | u = 1 \bmod n\}$.

### 6.4.3 Compute *argmax* over Ciphertext

After MLSP computes the inner products (i.e., encrypted inner product values) by taking as input its owned model and the encrypted feature vector from the user, it needs to figure out which ciphertext corresponds to the maximum value. Therefore, we propose a private comparison protocol which can preserve the data privacy while allowing MLSP to determine the maximum value, which is run between MLSP and the assisting cloud AS (note that we assume AS never colludes with MLSP) by leveraging the two-stage decryption technique discussed in Section 6.3.1.

**Comparison of two encrypted integers**. To be more specific, MLSP wants to compare two encrypted integers $[\![v_1]\!]$ and $[\![v_2]\!]$ of $l$ bits without revealing the real values, MLSP holds a private key share $x_1$ and AS holds the other private key share $x_2$ ($x_1$ and $x_2$ can be used to recover the user's private key $x$). We modify the comparison scheme proposed by Veugen [145], which is shown in Fig. 6.4. In our scheme, besides Paillier, we use another homomorphic cryptosystem over individual bits, namely Quadratic Residues (QR) [77]. Let $[\cdot]$ denote the ciphertext of the QR cryptosystem, and $z_{l+1}$ denote the $(l{+}1)$-th bit of integer $z$.

The basic idea of the comparison protocol is to use the significant bit of the value $\delta = v_1 + 2^l - v_2$, determining whether $v_1 > v_2$. Note that if the significant bit $\delta_{l+1}$ is 1, then $v_1 \geq v_2$, and $v_1 < v_2$ otherwise. Given that $\delta$ is a $(l+1)$-bit number (at most), its significant bit can be obtained through $\delta \div 2^l$ where $\div$ is the integer division, and $\delta = 2^l(\delta \div 2^l) + (\delta \bmod 2^l)$ where $0 \leq (\delta \bmod 2^l) < 2^l$. Moreover, in order to hide the difference between $v_1$ and $v_2$, let $\delta' = \delta + r$ where $r$ is a $l+1$-bit random value. Therefore,

$$\delta' = 2^l(\delta' \div 2^l) + (\delta' \bmod 2^l) = 2^l(\delta \div 2^l + r \div 2^l) + (\delta \bmod 2^l + r \bmod 2^l).$$

and $\delta' \div 2^l = \delta \div 2^l + r \div 2^l + t$ where $t = 0$ if $(\delta \bmod 2^l) + (r \bmod 2^l) < 2^l$, and $t = 1$ otherwise.

On the other hand, $t = 0$ means that $(\delta' \bmod 2^l)$ is equal to $(\delta \bmod 2^l) + (r \bmod 2^l)$. Therefore, if $(\delta' \bmod 2^l) \geq (r \bmod 2^l)$, then $t = 0$. Hence, the significant bit $\delta_l$ can be determined as follows:

$$\delta \div 2^l = (\delta' \div 2^l) - (r \div 2^l) - t \bmod 2 = z_{l+1} \oplus r_{l+1} \oplus t.$$

Based on the above idea (together with the two-stage decryption cryptosystem), we present the detailed comparison protocol as shown in Fig. 6.4. This comparison protocol corresponds to one interaction between MLSP and AS mentioned in Step 4 of Fig. 6.1. Note that in Step 8 of Fig. 6.4, MLSP and AS need to privately compare two integers, which is known as millionaires problem [164], and can be achieved by any garbled circuit scheme. Nevertheless, we implement it by applying a cryptography-based scheme, known as DGK [57], for the sake of compatibility to our cryptography-based framework.

Unlike existing work [57, 44, 154], where the comparison protocol was implemented with the help of the client holding the secret key through multiple-round interaction, our comparison pro-

MLSP **Input**: $[\![v_1]\!]$, $[\![v_2]\!]$, partial secret key $x_1$, and public QR key $PK_{QR}$
AS **Input** : partial secret key $x_2$, and private QR key $SK_{QR}$

1. MLSP: $[\![\delta]\!] \leftarrow [\![v_2]\!] \cdot [\![2^l]\!] \cdot [\![v_1]\!]^{-1} \mod N^2$ // $\delta \leftarrow v_2 + 2^l - v_1$
2. MLSP chooses a random value $r$ to blind $\delta$
3. MLSP: $[\![\delta']\!] \leftarrow [\![\delta]\!] \cdot [\![r]\!] \mod N^2$ // $\delta' \leftarrow \delta + r$
4. MLSP runs the algorithm Transform to partially decrypt $[\![\delta']\!]$ to $[\![\delta']\!]'$ with $x_1$, and sends $[\![\delta']\!]'$ to AS, where the Transform algorithm is defined as letting the transformed ciphertext $[\![\delta']\!]' = (c_1', c_2')$: $c_1' = c_1$ and $c_2' = c_2 / c_1^{x_1}$.
5. AS runs the algorithm FinalDec to decrypt $[\![\delta']\!]'$ with $x_2$: $\delta' = L(c_2' / c_1'^{x_2} \mod n^2)$.
6. MLSP: $c \leftarrow r \mod 2^l$
7. AS: $d \leftarrow \delta' \mod 2^l$.
8. MLSP and AS privately compute the encrypted bit $[t]$ such that $t = 1$ if $d < c$, and $t = 0$ otherwise.
9. AS encrypts $z_l$ using QR encryption and sends $[z_l]$ to MLSP.
10. MLSPencrypts $r_l$ to $[r_l]$.
11. Server computes $[t'] \leftarrow [\delta'_{l+1}] \cdot [r_{l+1}] \cdot [t]$ and sends $[t']$ to AS
    // $t' \leftarrow \delta'_{l+1} \oplus r_{l+1} \oplus t \equiv (t' = \delta \div 2^l)$
12. AS decrypts to get $t'$

Figure 6.4: Comparison protocol of two encrypted integers

tocol reduces the heavy computational overhead without having the user engaged in the protocol, while leveraging the assisting cloud AS and the two-stage decryption cryptosystem.

**Classification result permutation**. Given the comparison protocol for two encrypted integers as above, we extend it to a complete protocol implementing *argmax*. An intuitive way is that AS sends the comparison result back to MLSP so that the comparison process can be repeated until finding the greatest one, incurring linear complexity (i.e., $\mathcal{O}(k)$). This approach, however, violates the security goals as mentioned in Section 6.2.3, because each comparison reveals the greater one

MLSP **Input** : $k$ encrypted integers $[\![v_1]\!], \cdots, [\![v_k]\!]$, partial secret key $x_1$, and public QR key $PK_{QR}$
AS **Input** : partial secret key $x_2$, and private QR key $SK_{QR}$
**User Output**: classification result $i$

1. MLSP: chooses a random permutation $\pi$ over $\{1, \cdots, k\}$, re-indexes the $k$ values, and sends the permutation map to the user.
2. MLSP and AS perform the following iteration to get index $s$ that leads to the greatest $[\![v]\!]$:

   MLSP: Let $[\![max]\!] = [\![v_{\pi(1)}]\!]$
   AS: Let $[\![max]\!] = [\![v_{\pi(1)}]\!]$ and $s = 1$

```
    for $i = 2$ to $k$ do
        Server↔AS: interact to compare $[\![max]\!]$ and $[\![v_{\pi(i)}]\!]$ using the comparison protocol in Figure 6.4
        if $[\![max]\!] < [\![v_{\pi(i)}]\!]$ then
            $[\![max]\!] = [\![v_{\pi(i)}]\!]$
            $s = i$
        end
        AS: randomizes $[\![max]\!]$ as $[\![max]\!] = [\![max]\!] \cdot [\![0]\!]$ and sends $[\![max]\!]$ back to MLSP
        //max = max + 0
    end
```

3. AS: sends $s$ to client
4. User: gets the classification result $\pi^{-1}(s)$

Figure 6.5: *argmax* protocol

to MLSP, which therefore eventually leaks the classification result. Another alternative is that MLSP and AS compute all two-element combinations of the $k$ values $[\![v]\!]$. By doing this, AS will learn which one is the greatest, but MLSP knows nothing, which then achieves the security goal. Unfortunately, this approach incurs $\mathcal{O}(k^2)$ complexity. In addition, since AS is semi-honest, and it should not know the final classification result as well.

To overcome the drawbacks of the above two approaches, we propose a three-party protocol by applying permutation, which is depicted in Fig. 6.5, which incurs only linear complexity $\mathcal{O}(n)$ and hides the classification result from MLSP and AS. The basic idea works as follows (referred to Step 3, 5, 6 of Fig 6.1): MLSP applies a random permutation $\pi$ over $k$ values $[\![v]\!]$ such that the $i$-th value becomes the $\pi(i)$-th value, and then sends the permutation map to the client; then MLSP and AS iteratively perform the comparison protocol (Fig. 6.4) until the index $s$ that leads to the greatest $[\![v]\!]$ after $k-1$ iterations. More specifically, at each iteration, MLSP compares the current maximal value $[\![max]\!]$ to the next value with the help from AS. Once the maximum of the two compared values is determined, to prevent MLSP from linking the maximum to the value compared, AS randomizes it by adding an encrypted $[\![0]\!]$ to $[\![max]\!]$. Since Paillier encryption allows adding randomness to the ciphertext, the same $[\![max]\!]$ blinded with different encryptions of 0 looks like different random numbers to MLSP. Hence, MLSP does not know which value is greater in the comparison. When comparisons finish, AS sends the index $s$ that leads to the greatest value to the user, who in turn gets the classification result by reversely mapping $s$ to the class. Eventually, MLSP knows nothing about the order of the $k$ values. Although AS learns that the $s$-th value is the greatest, it cannot correlate it to a specific class due to the permutation. Therefore, the classification result is hidden from MLSP and AS as we expect. Note that in the *argmax* protocol, the user only receives a mapping (generated by the permutation $\pi$) and does a very lightweight lookup, and which bring very little communication and computation overhead to the user.

### 6.4.4 Discussion

To make the classification over encrypted data feasible for IoT devices, we outsource most part of the computation and communication to an assisting cloud AS. However, we need to clarify that, if we consider all participants including client, MLSP and AS as a whole, our three-party scheme does not reduce the whole complexity of computation and communication. On the contrary, since we adopt two-decryption Paillier scheme, the computation cost on MLSP may be increased. And thus, it is not guaranteed that the time spent on an entire process of one-time classification service from initiating service request by client to receiving the classification result is shorter. The improvement on shortening the time of the entire classification process will rely on multiple factors, such as the load and job scheduling on the cloud, the hardware of instances used on MLSP and

AS, and the network latency between them, etc. We want to make it clear that, like many other cloud-assisted schemes, the goal of introducing an assisting cloud is to shift the computation and communication from one device to the cloud, particularly from a resource-constrained device, but such a shifting is not straightforward especially with privacy constraint. Our contribution lies in the way how we realize the privacy-preserving shifting. Without that shifting, some applications may be infeasible. For example, if energy-constrained wearable devices use the costly two-party scheme, the energy will be used up very soon. Nevertheless, with our cloud-assisted scheme, although it may not always speed up the classification process, it can definitely reduce the overhead on wearable devices and make them last longer.

## 6.5 Security Analysis

In this section, we analyze the security of the proposed scheme in terms of input data privacy, machine learning model privacy and classification result privacy. Note that in our model, the three parties (i.e., the user, AS and MLSP) cannot collude with each other.

**Input data privacy.** When computing the inner product protocol as shown in Fig. 6.3, the feature vector is encrypted by the user's public key. Note that either MLSP or AS only possesses a share of the secret key, they cannot decrypt the encrypted data as we assume that both MLSP and AS cannot collude together. In addition, in the comparison protocol as shown in Fig. 6.4, AS blinds the inner product with a random value (i.e., $r$ in Step 3), so that MLSP cannot know the inner product without knowing that random value. Furthermore, with the privacy-preserving integer comparison protocol (i.e., DGK protocol in our proposed scheme), AS only knows which inner product value is the greatest but not knowing its exact value (only learning the index of the classification result). That is, either AS or MLSP cannot learn either the input data or the inner product values.

**Machine learning model privacy.** Note that in the inner product and comparison protocols, because the comparison protocol (Fig. 6.4) is implemented in a privacy-preserving way, either the user or AS cannot know the inner product values with the given feature vectors. That is, the user and AS has no chance to learn the machine learning model.

**Classification result privacy.** MLSP permutes the indexes of the classes so that the permutation mapping is only shared between the user and MLSP. Moreover, AS only knows the permuted index for the classification result (through the comparison protocol). Therefore, MLSP has no knowledge about the classification result. Only the user can get the classification result by using the permuted index and the permuted mapping. Therefore, only the user can know the classification result while the other two parties are blind to that.

## 6.6 Performance Evaluation

We have implemented the proposed cloud-assisted machine learning service scheme in real-world clouds, i.e., Amazon EC2, and compared the performance of our scheme with existing works [44, 43] in terms of asymptotic complexity and the experimental performance. More specifically, we compare our work to Bost *et al.*'s scheme [44], which adopts lightweight cryptographic primitives to protect the user's input data and MLSP's model mutually in a two-party setting. Note that we did not compare our proposed scheme with that in [43], which uses fully homomorphic encryption (FHE) to allow the server to compute some medical predication functions over patient's encrypted input. The reason is that, on the one hand, FHE is still far more practical and will introduce significant overhead, and on the other hand, [43] only protects the user's input and assumes that MLSP's model is known to the public, thereby achieving weaker privacy guarantee when compared to our work.

When conducting the performance comparison, we focus on the computational overhead at the user side, due to the fact that our goal is to minimize the cost on the recourse-constrained IoT devices and it is commonly assumed that MLSP and AS have unlimited computational capability.

### 6.6.1 Asymptotic Complexity Analysis

Table 6.1 shows the asymptotic complexity comparison of the two schemes at the user side, where $k$ denotes the number of classification classes, $d$ denotes the dimension of the feature vector, Exp de-

| Scheme | Inner Product | Argmax |
|---|---|---|
| Ours | $d \times \mathsf{Enc}$ | $\mathsf{Dec}$ |
| Bost's | $k \times d \times (\mathsf{Exp} + \mathsf{Add})$ | $k \times (5\mathsf{Add} + \mathsf{Exp} + 2\mathsf{Enc} + \mathsf{DGK})$ |

Table 6.1: Comparison of asymptotic complexity at the user side between our proposed scheme and Bost's [44].

| Scheme | Number of messages |
|---|---|
| Ours | $(d+1)\times$ ciphertext + Permutation-Map |
| Bost's | $(d+k\times(3+2l))\times$ ciphertext |

Table 6.2: Comparison of communication overhead on the client of two schemes in terms of the number of messages.

notes the exponentiation operation, DGK denotes the overhead on the user side when executing the DGK protocol used in argmax protocol, Enc, Dec, Add denote encryption, decryption and adding two ciphertexts, respectively. We can see that when comparing with that of [44], our proposed scheme greatly offloads the computational overhead on the user side.

In addition, we also compare the communication overhead for the user. In our scheme, it is easy to see that the communication overhead only comprises $d$ ciphertexts as the user's input, a permutation map and a ciphertext containing the classification result. Note that as in our scheme, all communication involved in the argmax protocol has been offloaded to AS and therefore save the communication cost for the user. In contrast, the user in [44] engages in both the inner product protocol and argmax protocol, and incurs significant communication overhead. We detailed the communication overhead comparison on the user in Table 6.2 in terms of the number of messages because in our scheme the size of permutation map (which is sent to the user) is much less significant than that of ciphertexts to be transfered, which dominates the communication overhead complexity. From Table 6.2, we can see that the communication overhead on the user side in our scheme is much less than that of the Bost's scheme.

### 6.6.2 Implementation and Performance Evaluation

We have implemented the Bost's scheme and our proposed scheme with Java where three cryptographic primitives are used, i.e., the original Paillier, the revised Paillier and the Goldwasser-Micali (QR) cryptosystems. In our implementation, we set the bit length of the large prime numbers $p$ and $q$ used throughout three cryptosystems to a strong security level, i.e., 512, so $n$ is 1024-bit length. The experiments are conducted on the Raspberry Pi 2, with 700MHz ARM A6 microprocessor and 512MB RAM, to simulate the resource constrained IoT devices such as smart phone acting as the gateway for wearable devices and smart meters.

Fig. 6.6 compares the computation time spent on the client finishing a classification service with different parameter settings. Since the performance goal of this work is to mitigate the overhead on resource-constrained IoT device, we measure the time spent on all computational operations which are performed by the client. Theses computational operations include encryption of feature vector, multiple-round comparison of encrypted integers and decryption of classification result, if any of above operation is applicable in the compared schemes. Note that the time is not the entire time from initiating the service request to receiving the classification result, which excludes the time of network communication and processing time on MLSP and AS. For example, in the setting where there are 10 classes and 10 features, and the key length is 1024, we can see in Fig 6.6, the overhead on user in Bost's two-party scheme [44] is almost 200 times than our three-party scheme. Our asymptotic complexity analysis is confirmed by the real implementation, that is, the overhead on the client is much more lightweight than Bost's scheme, since our scheme outsources the costly operations to the cloud. It is worth noting that the overhead on the client in Bost's scheme is proportional to the number of classes, while the client's overhead in our scheme is independent from the number of classes but proportional to the number of features due to the encryption of features.

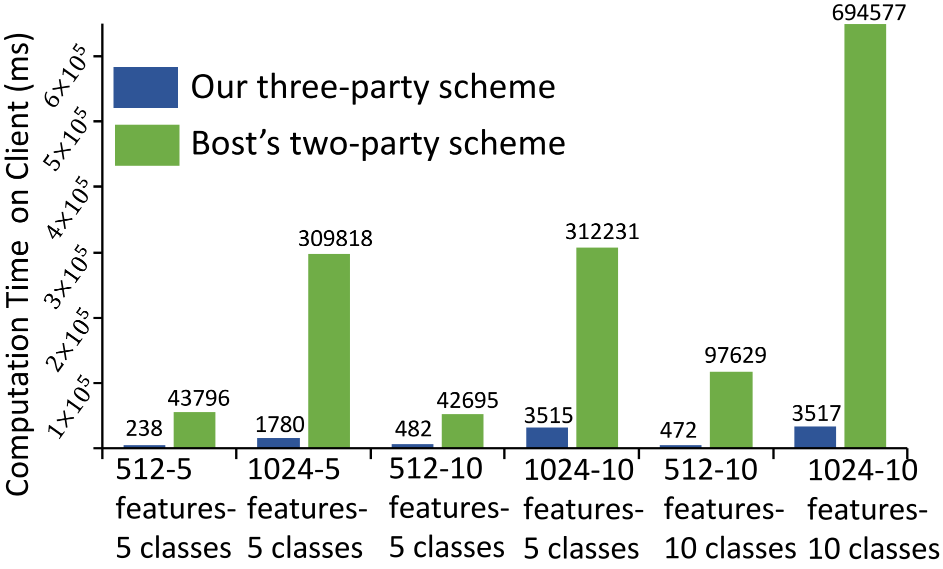


Figure 6.6: Comparison of overhead on client between our scheme and Bost's [44] with varying key length, number of features and number of classes. The Y-axis is the computation time on client, which includes the time of all computational operations in process of finishing one-time classification service, such as encryption of feature vector, comparison of encrypted integers, and decryption of the classification result (if applicable), but it excludes the time on network communication and processing on MLSP and AS.

## 6.7 Related Work

All related work to this paper can be roughly classified into three categories:

**Privacy-preserving training**. Training the model is the first phase of a complete machine learning process. Most existing work falls into this category, which either use cryptographic techniques [107, 143] or privacy-preserving data mining techniques such as value distortion [26], randomized response [61] and partitioned data [144, 63]. These work span over variety of training algorithms including logistic regression [50], decision trees [144, 63, 61], clustering [143], Naive Bayes [153], etc.

**Privacy-preserving classification**. Little work has been done in the category of privacy-preserving classification, namely, apply the trained model to client input. Our paper falls in this category. Erkin *et al.* proposed a privacy-preserving face recognition scheme to hide biometrics from the server conducting the matching operation [64]. Bos *et al.* proposed a fully homomorphic encryption (FHE) based scheme to allow third party to perform predication over the encrypted medical record of a patient [43]. However, the performance problem of FHE makes their scheme less practical in reality. Besides, in their work, the server's model is assumed to be known by the public, so they provide no protection to the predictive model, and thus weaker security guarantee than our scheme. Wu *et al.* proposed a protocol to privately evaluate decision trees [154] using DGK comparison protocol [57], oblivious transfer and tree permutation techniques. In their scheme, the client's input and the server's tree model are mutually hidden from the opposite party. In contrast to the previous schemes that focus on a specific learning algorithm, Bost *et al.* proposed a set of building blocks to construct more complex classifiers, and meanwhile they also protect the information of client and server simultaneously [44]. However, to make their scheme support more classification algorithms such as decision trees besides hyper-plane based classifiers focused in out paper, they view the decision tree as a polynomial and adopt FHE to compute the polynomial, which inevitably results in worse performance. More importantly, all the above schemes work in a two-party client-server model and did not pay special attention to the fact that the client may have very limited capacity of computation and communication, such as the IoT devices on which we focus.

**Cloud-assisted IoT security**. A broader topic related to our work is cloud-assisted IoT security. Since IoT sensors such as wearable devices, smart meters, in-home monitoring cameras usually collect and report sensitive data, and plenty of application use the data in various ways, to keep the data secure, cryptographic techniques are often an alternative option. However, resource-constrained IoT devices usually cannot afford costly cryptographic techniques and large data storage. Many schemes are solving this problem by leveraging cloud which provides unlimited computation and storage capacity [166, 83, 157, 163]. Zhou *et al.* proposed a privacy-preserving key management scheme for cloud-assisted wireless body area networks where the computationally-intensive key material updating is outsourced to the cloud in a privacy-preserving way [166]. [83, 163] both focus on cloud-assisted healthcare IoT, which mainly use the storage resources of the cloud. In [83], they proposed a scheme to add watermark into the collected data of a patient to avoid the privacy leakage on the cloud, while Yang *et al.* proposed a scheme that allows health service providers such as doctors to access and verify the encrypted medical records stored on the cloud by using a searchable encryption with forward privacy support [163]. In contrast, [157] utilizes the computation resources of the cloud to implement a data publishing scheme adopting attribute-based encryption.

## 6.8 Summary

In this chapter, we proposed a cloud-assisted, privacy-preserving machine learning classification scheme for resource-constrained IoT devices. By introducing an additional cloud server and employing a two-stage decryption Paillier-based cryptosystem, our scheme allows an IoT device to offload expensive classification computations to the cloud server in privacy-preserving manner, thereby ensuring data privacy for both IoT client and machine learning service provider. The extensive complexity analysis and performance evaluation demonstrate that the proposed scheme provides an efficient solution for conducting machine learning on IoT devices, where compared to the existing solutions in the literature.

# Chapter 7

# A Multi-Cloud based Privacy-Preserving Data Publishing Scheme for the Internet of Things

## 7.1 Introduction

In Part II, we aims to secure the data and privacy for IoT applications and we have proposed two schemes for privacy-preserving data storage and privacy-preserving data analysis in Chapter 5 and Chapter 6, respectively. In this chapter, we will explore the next problem, that is, targeted broadcast in IoT.

Recall that the charm of IoT lies in its capability of interconnecting various sensing devices with varying computation and communication capacity to the Internet. For example, connecting millions of sensors and smart meters to the utilities and power plants makes the aging power grid "smart". The two-way communication capability not only facilitates utilities to collect a variety of real-time, fine-grained data (e.g., power consumption, voltage, phase angle) from smart meters, but also enables them to publish important messages (e.g., control commands, dynamic prices) to smart devices [114]. Similarly, the success of wearable devices and e-health depends not only on the real-time data sensing and reporting, but also on the timely reaction (e.g., firmware upgrade, parameter adjustment) and value-added services (e.g., customized healthy living tips, targeted advertisements) from healthcare service providers and third parties [21]. For example, consider the high risk of heart attacks in extremely hot weather, it is a desirable feature for a wearable device manufacturer to send control messages to devices to increase the measurement frequency of blood pressure and oxygen saturation for elder users with heart diseases and overweight issues.

Nevertheless, with all devices connected and using the Internet infrastructure for data exchange, IoT and its two-way communication expose a new attack surface to adversaries, which makes it susceptible to various security and privacy issues. Consider the tremendous loss due to system failures and attacks, such as power outage and patient data breach, security and privacy has become one of the most important aspects in the design and deployment of IoT applications. Various schemes have been proposed to secure IoT communication and data exchange, which span over a variety of topics including but not limited to privacy-preserving data collection [102, 111], detection and prevention of false data injection [101, 110, 80, 158], patient monitoring [127, 112], etc. However, most of the attention has been concentrated on secure data collection, which considers only one direction (i.e., the *upward* link) of the two-way communication. Little work has been done to secure the other direction (i.e., the *downward* link), where messages are pushed to millions of end devices, neglecting the fact that data along this direction sometimes contains sensitive information, such as system parameters, user-specific prices and healthcare information, and thus needs an enhanced protection.

Conventionally, to securely publish data to a specific target group, the sender (e.g., a control center in smart grid or a value-added service provider in an e-health system) sends data in an encrypted form to each device in a unicast manner. Given the huge number of devices in IoT applications, this will incur a large communication and computation overhead [79] as well as a high complexity for key management. To alleviate the overhead, approaches based on broadcast encryption have been proposed to encrypt the message for an arbitrary set of receivers so that only members of the target group can recover its content. However, solutions based on conventional broadcast encryption [67, 42] require the sender to store key materials for all recipients and incur a storage requirement for the sender. This makes them impractical for IoT applications with a huge number of receiving devices. Besides, in the conventional broadcast encryption, all members in the broadcast set need to be notified of the change [42], which incurs a non-negligible communication overhead to support the incremental growth of the number of users. Therefore, neither option is particularly appealing in the context of IoT.

A more promising solution is to encrypt the message according to common characteristics that specify a set of users as the qualified recipients. For example, recipients can be categorized by their geographical locations such as "*users in San Francisco*". Bethencourt et al. proposed the concept of ciphertext policy (CP-ABE) [37], which is conceptually similar to role-based access control but more fine-grained and flexible. In the CP-ABE model, a user is associated with a set of attributes reflected in her secret key and the access policies, which define the attributes that an authorized user should own, are embedded in the ciphertext. The sender encrypts a message under an access policy so that only authorized users with attributes satisfying the access policy can decrypt the message. The ABE schemes enable a sender to define dynamic access policies without knowing specific receivers in the system beforehand, and thus suit the IoT scenario to enforce fine-grained access control on a large number of receivers. However, a major drawback of ABE is the computational cost for decryption, which increases linearly with the complexity of the access policy (i.e., the number of attributes), due to the expensive paring operations when the receiver matches her attributes to the access policy. As tested in [79], it took about 30 seconds for a smart phone (in particular an iPhone 3G in that experiment) to decrypt a ciphertext containing 100 attributes. For resource-constrained recipients that are typical in an IoT application, the computational cost is too expensive to afford. With the emergence of cloud computing infrastructures such as Amazon EC2 and Microsoft Azure, outsourced-ABE schemes have been proposed [79, 167, 87, 103] to leverage the computational power of the clouds by outsourcing a part of the expensive decryption operations to them.

However, privacy becomes a critical concern when outsourcing decryption operations to a cloud. Most of the existing outsourced-ABE approaches assume that the cloud is fully trusted to host all attributes of a user. In the real world, a cloud server is rarely fully trusted by both the sender and the recipient, especially when user attributes involve multiple sensitive categories. For example, a cloud that is trusted to process attributes such as "*age*" and "*location*" may not be able to access attributes related to users' health status, such as "*having heart disease X*" and "*weight>Y*". Therefore, it is more reasonable to assume a cloud as *honest-but-curious*, that is, it follows the protocol honestly but tries its best to infer users' private information. Specifically, there are three major privacy concerns: (1) *Data privacy.* The sender, e.g., value-added service providers, requires its messages and services to be only accessible to a group of qualified users. Neither the cloud nor other unauthorized users should be able to access the published data. (2) *Attribute privacy.* As explained in the above example, attributes contain sensitive information about the user and thus should be protected as much as possible from being disclosed to the cloud. And (3) *Access policy privacy.* Access policies containing information about data to be published, data sender and data recipient also need to be protected. A stronger privacy requirement is to hide the access policy to avoid privacy inference attacks. We argue that the importance of three types of privacy can be ranked as: data privacy > attribute privacy > access policy privacy.

In general, outsourcing the decryption operations to the cloud is a promising technique to publish information to a group of resource-constrained devices in IoT applications. However, protecting the three types of private information in the targeted broadcasting is still challenging. To address the privacy issues discussed above, we propose two novel CP-ABE schemes that employ multiple clouds to collaboratively complete the outsourced operations, namely *parallel-cloud CP-ABE* and *chain-cloud CP-ABE*. Our schemes delegate user attributes and the decryption operations to multiple clouds, and coordinate them to translate an ABE ciphertext into an ElGamal-type ciphertext, without revealing the original message, the accurate set of user attributes, or the complete access policy to the clouds.

Besides protecting the three types of privacy, our schemes also provide two additional features that are desirable in the IoT data publishing scenario: (i) *Delegation verifiability.* When we outsource the operations for matching attributes to the access policy to the cloud, it is important for the receiver to be able to verify the correctness and completeness of the messages processed by the cloud. For example, a “cheating” cloud server may violate the honest-but-curious assumption and discard messages without performing the matching operation to save its own communication and computation resources. Therefore, the delegation verifiability can be considered as a security enhancement for IoT data publishing applications. And (ii) *Lightweight operations on receivers.* Since the receivers are resource-constrained devices, the operations of decryption and verification at the receiver end are expected to be kept light-weight, which is supported in our solution.

The main contributions of this work are as follows:

- To the best of our knowledge, our work is among the first to protect user attributes against cloud service providers in an outsourced-ABE setting using multiple clouds.
- We propose two multi-cloud-based, outsourced-ABE schemes. The parallel-cloud scheme provides a better performance with strong privacy protection but less system flexibility and less expressiveness for access policies, while the chain-cloud scheme supports flexible customization and expressive access policies at the cost of processing latency.
- We present a lightweight mechanism that allows users to efficiently verify the correctness and completeness of the partial decryption in clouds.
- Our schemes enable a new application for data publishing in IoT, namely privacy-preserving targeted broadcasting, which is not possible in the past. This new type of data publishing application will benefit not only end users but also information senders such as third-party value-added services providers.
- We thoroughly analyze the security of our proposed schemes and evaluate the performance with experiments using Amazon EC2 and Windows Azure.

## 7.2 Preliminaries

### 7.2.1 Bilinear Maps

Let $\mathbb{G}_0$ and $\mathbb{G}_1$ be two multiplicative cyclic groups of prime order $p$, and $g$ be a generator of $\mathbb{G}_0$. $e$ is a bilinear map $e : \mathbb{G}_0 \times \mathbb{G}_0 \rightarrow \mathbb{G}_1$ with the following properties:

1. Bilinearity: $e(u^a, v^b) = e(u,v)^{ab}$ for all $u, v \in \mathbb{G}_0$ and $a, b \in \mathbb{Z}_p$.

2. No-degeneracy: $e(g,g) \neq 1$.

3. Computability: for all $u, v \in \mathbb{G}_0$, the bilinear map $e$ is efficiently computable.

### 7.2.2 Linear Secret Sharing Scheme (LSSS)

A ($k$,$n$)-LSSS shares a secret $s$ over a set of $n$ parties with linear reconstruction property. The secret $s$ is divided to $n$ parties in such a way that any $k$ or more parties can recover the secret and any $k-1$ or less leave the secret completely undermined. Specifically, Shamir's secret-sharing scheme [129] is constructed as:

1. Pick $k-1$ random points to define a polynomial $q(x)$ of degree $k-1$ with $q(0) = s$.

2. Share the secret over $n$ parties by computing $q(i)$ for any $i \in \{1, \cdots, n\}$.

3. Reconstruct the polynomial from any $k$ parties denoted by a set $S$, using interpolation $q(x) = \sum_{i=1}^{k} q(i)\Delta_{i,S(x)}$. The Lagrange coefficient is $\Delta_{i,S(x)} = \prod_{j\in S, j\neq i} \frac{x-j}{i-j}$, where $S(x)$ denotes an element of $S$. Finally, recover the secret $s = q(0)$.

### 7.2.3 Bloom Filter

Bloom filter (BF) [39] is a space-efficient probabilistic data structure for an approximate representation of a set $S$, which is typically implemented using a bit-array of $w$ bits with $k$ hash functions. Given an arbitrary element $x$, a BF supports approximate membership queries "$x \in S$?". It can

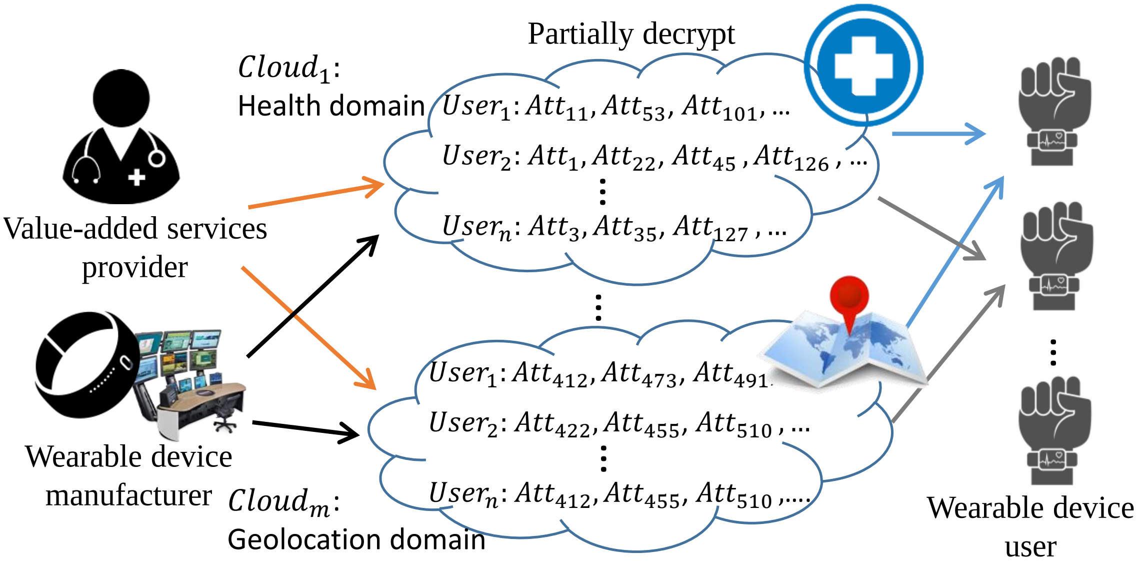

Partially decrypt
$Cloud_1$:
Health domain
$User_1$: $Att_{11}, Att_{53}, Att_{101}, \ldots$
$User_2$: $Att_1, Att_{22}, Att_{45}, Att_{126}, \ldots$
$User_n$: $Att_3, Att_{35}, Att_{127}, \ldots$
Value-added services provider
Wearable device manufacturer
$User_1$: $Att_{412}, Att_{473}, Att_{491}$
$User_2$: $Att_{422}, Att_{455}, Att_{510}, \ldots$
$User_n$: $Att_{412}, Att_{455}, Att_{510}, \ldots$
$Cloud_m$:
Geolocation domain
Wearable device user

Figure 7.1: Framework of privacy-preserving targeted broadcast in e-healthcare using parallel-cloud scheme.

yield false positive answers but never false negative ones. The probability of false positives can be adjusted by varying $w$ and $k$, as a tradeoff between space efficiency and the false positive rate.

## 7.3 Parallel-cloud Scheme

Existing outsourced ABE schemes assume the cloud provider is fully trusted and thus delegate attributes to a single cloud server. However, under the honest-but-curious setting, this assumption is no longer valid. To prevent the cloud server from inferring user privacy from outsourced keys and attributes, we propose a new secure data publishing approach that employs multiple non-colluding cloud servers. We first present a parallel-cloud scheme that divides the attribute set into $m$ parts and outsources each part to one cloud server. The cloud servers operate on the received access structure and ciphertext messages in parallel, and send the intermediate results to the receiver separately. In this process, as long as the cloud servers do not collude with each other, we can protect the complete set of user attributes from any single semi-honest cloud server.

### 7.3.1 System Model and Scheme Overview

In this scheme, the system consists of three entities, as shown in Figure 7.1: the *sender*, such as a value-added service provider or a wearable device manufacturer, publishing a message in the encrypted form; *m cloud servers* partially decrypting the ciphertext; and a large set of targeted device *users* receiving and decrypting the message. In addition, a trusted authority (TA) is implicitly assumed to be in charge of the distribution and management of attributes and private keys to users and cloud servers.

To enforce the fine-grained broadcasting, the sender encrypts a message according to a specific access policy $\mathbb{A}$ in the form of a series of AND gates. For instance, an access structure $\mathbb{A} = \mathit{DeviceA} \wedge \mathit{HeartDisease} \wedge \mathit{Overweight} \wedge \mathit{CityB}$ defines that only the user living in City B who is a customer wearing device A with the heart disease and the overweight issue can decrypt the message. For the simplicity of exposition, we let $N$ denote the universal attribute set of $n$ attributes. The TA divides $N$ into $m$ mutually exclusive subsets as $N = N_1 \cup \cdots \cup N_m$, where $N_i \cap N_j = \emptyset$ for any $i \neq j$, and then outsources each $N_i$ to a cloud server.

For privacy preserving considerations, the attribute splitting and distribution should follow two strategies. First, each subset of attributes is only about one aspect (or domain) of the user. Attributes in any single subset should not provide information for inference attacks. Secondly, the attribute subset is assigned to a cloud considering its service domain and trust level. For example, the cloud server of a private hospital is trusted to host all health-related attributes, while common user attributes such as location can be distributed to a public cloud server.

Correspondingly, the access structure is also divided into $m$ parts such that $\mathbb{A} = \mathbb{A}_1 \cup \cdots \cup \mathbb{A}_m$. When the sender sends the main ciphertext and access structures to cloud servers, each cloud server, on behalf of the receiver, checks if the attributes of the receiver satisfy the access structure. Based on the result of this fine-grained access control, each cloud server decrypts a part of the message and sends it to the user, who will combine the intermediate results received from all the servers to recover the original message.

### 7.3.2 Construction

**Setup($\lambda$, $N$, $m$).** The TA calls the Setup algorithm to generate a public key $PK$ and a master key $MK$. The algorithm takes as input the security parameter $\lambda$, a universal attribute set $N$ and the number of cloud servers $m$. It chooses a bilinear group $\mathbb{G}_0$ of prime order $p$ with a generator $g$, and the bilinear map $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$. Next, the setup algorithm chooses two randoms $\alpha, \beta \in \mathbb{Z}_p$ and generates $PK$ and $MK$:

$$PK = (\mathbb{G}_0, g, h = g^{\beta}, e(g,g)^{\alpha}),$$
$$MK = (\beta, g^{\alpha}).$$

**KeyGen($MK$, $S$, $t$).** Assume each user has a set of attributes $S = S_1 \cup \cdots \cup S_m$, where $S_1 \subseteq N_1, \cdots, S_m \subseteq N_m$ and $\forall_{i \neq j} S_i \cap S_j = \emptyset$. When a new user (e.g., a wearable device) joins the system, it registers to the TA with $S$ and a random $t \in \mathbb{Z}_p$ chosen by itself. Then, the TA calls the key generation algorithm to prepare a transformation key $TK$ for the clouds to perform partial decryption, from which the user can recover the final message with his private key $SK = t$. In particular, KeyGen selects a random $r \in \mathbb{Z}_p$ and a random $r_j \in \mathbb{Z}_p$ for each attribute $j \in S$, and takes $S$, $t$, and the master key $MK$ as input to generate the transformation key. $TK$ is set as $TK = (D = g^{t(\alpha+r)/\beta}, D_j = g^r H(j)^{r_j}, D_j{}' = g^{r_j}, \forall j \in S)$. $H$ and $H_1$ denote collision-free hash functions, where $H : \{0,1\}^* \to \mathbb{G}_0$. Finally, the TA distributes $D$, $D_j$, and $D'_j$ to $Cloud_j$.

**Encrypt($PK$, $M$, $\mathbb{A}$).** To broadcast a message $M$ under the access structure $\mathbb{A}$, the sender first chooses $m$ random numbers $s_1, \cdots, s_m \in \mathbb{Z}_p$, where the secret $s_i$ is shared by all attributes in $\mathbb{A}_i$. Let $k_i = |\mathbb{A}_i|$ be the number of elements in $\mathbb{A}_i$ and $index(y)$ be the index of the attribute $y$ in $\mathbb{A}_i$. To share the secret, a polynomial $q_i(x)$ of degree $k_i - 1$ is constructed for $\mathbb{A}_i$, where $q_i(0) = s_i$ and the other $k_i - 1$ values are randomly set to complete the construction. Given $\mathbb{A}_i$, the ciphertext $CT$ is then constructed as:

$$\widetilde{C} = Me(g,g)^{\alpha s}, C = h^s, CT_i$$

where $s = \sum s_i$, and $CT_i =$

$$\mathbb{A}_i, C_y = g^{q_i(index(y))}, C'_y = H(y)^{q_i(index(y))}, \forall y \in \mathbb{A}_i$$

To allow the receiver to verify the correctness and completeness of the transformation performed by the clouds, the sender also generates a verification value $V = senderID|\, H_1(\mathbb{A})|seq(H_1(\mathbb{A}))|T$, where "|" denotes concatenation. $H_1(\mathbb{A})$ is the digest of the access policy $\mathbb{A}$, $seq(H_1(\mathbb{A}))$ is the sequence number of the message regarding the access policy $\mathbb{A}$, and $T$ is the timestamp of the message. The sender maintains the sequence number for each access policy, and increases its value by 1 when sending a new message under that access policy. Then, the sender generates the signature of message $M$ and verification value $V$ as $\sigma = sign(H_1(M|V))$. Finally, the sender randomly chooses a cloud, e.g., $Cloud_1$, to hold $(\widetilde{C}, C, \sigma)$, and sends $CT_i$ and $V$ to $Cloud_i$.

**Transform($CT_i$, $TK_i$).** When a cloud, e.g., $Cloud_i$, receives $CT_i$, it uses the transformation key $TK_i$ to partially decrypt the ciphertext and transforms it into a form whose decryption is less computationally costly. In particular, the cloud first checks if $\mathbb{A}_i \subseteq S_i$. If not, it returns an error symbol $\perp$, indicating that the user does not satisfy the access structure. If $\mathbb{A}_i \subseteq S_i$, for each attribute $j \in \mathbb{A}_i$, it computes:

$$\begin{aligned} DecryptNode(CT_i, TK_i, j) &= \frac{e(D_j, C_j)}{e(D'_j, C'_j)} \\ &= \frac{e(g^r H(j)^{r_j}, g^{q_i(index(j))})}{e(g^{r_j}, H(j)^{q_i(index(j))})} \\ &= e(g,g)^{rq_i(index(j))} \end{aligned}$$

After the cloud computes the values for all attributes in $\mathbb{A}_i$, it combines them to partially recover the secret $s_i$ that is shared in $\mathbb{A}_i$. In particular, it computes $F(\mathbb{A}_i)$ as below and sends the result together with $V$ to the receiver:

$$\begin{aligned} F(\mathbb{A}_i) &= \prod_{j\in\mathbb{A}_i} (e(g,g)^{rq_i(index(j))})^{\Delta_{j,\mathbb{A}_i}(0)} \\ &= e(g,g)^{r\sum_{j\in\mathbb{A}_i} q_i(index(j))\cdot\Delta_{j,\mathbb{A}_i}(0)} \\ &= e(g,g)^{rq_i(0)} \\ &= e(g,g)^{rs_i} \end{aligned}$$

With $C$ and $D$, $Cloud_1$ computes $\widetilde{D}$ as:

$$\begin{aligned} \widetilde{D} &= e(C,D) \\ &= e(g^{\beta s}, g^{t(\alpha+r)/\beta}) \\ &= e(g,g)^{st(\alpha+r)} \end{aligned}$$

Finally, it sends the partially decrypted ciphertext $\widetilde{C}$, $\widetilde{D}$ and $\sigma$ to the receiver.

**Decrypt($\widetilde{C}, \widetilde{D}, F(\mathbb{A}_i), SK$).** If the receiver receives $m$ parts of partial ciphertexts, she knows that her attributes satisfy the access policy. Otherwise, she discards the partial ciphertext without decryption.

With its private key $SK$ and the ciphertext transformed by the clouds (i.e., $\widetilde{D}$ and $F(\mathbb{A}_i)$s), the receiver recovers the original message as:

$$\begin{aligned} \frac{\widetilde{C}}{\frac{(\widetilde{D})^{1/t}}{\prod_{i=1}^{m} F(\mathbb{A}_i)}} &= \frac{\widetilde{C}}{\frac{(e(g,g)^{st(\alpha+r)})^{1/t}}{\prod_{i=1}^{m} e(g,g)^{rs_i}}} && = \frac{\widetilde{C}}{\frac{e(g,g)^{s(\alpha+r)}}{e(g,g)^{rs}}} \\ &= \frac{Me(g,g)^{\alpha s}}{e(g,g)^{\alpha s}} && = M. \end{aligned}$$

In the above computation, it is obvious that the receiver does not need any paring operation to recover the message. Instead, it only takes one exponentiation and $m$ multiplication operations regardless of the complexity of the access structure. Compared to the original CP-ABE, which requires $2\sum|\mathbb{A}_i|$ pairings, our scheme greatly reduces the computational overhead at the receiver.

**Verify($M, \sigma$, V).** To verify the completeness of the operations done by the clouds, the receiver needs to check if all the verification values are consistent. As long as at least one cloud is honest, the received verification data is authentic. Then, the receiver randomly chooses a $V$ from $m$ partial ciphertexts, and uses *senderID* and $H_1(\mathbb{A})$ to look up the sequence number $seq_{prv}(H_1(\mathbb{A}))$ of the previous round for $H_1(\mathbb{A})$ from *senderID*. The receiver verifies the completeness, that is, she has received all messages intended to her, by checking if $seq(H_1(\mathbb{A})) = seq_{prv}(H_1(\mathbb{A})) + 1$. If not, this indicates some messages are discarded by dishonest cloud(s). After the completeness verification, the receiver checks $M|V$ against the signature $\sigma$ to verify the correctness of the recovered message, and updates $seq(H_1(\mathbb{A}))$.

**Discussions.** The parallel-cloud scheme hides the complete set of user attributes from a single cloud server, and significantly reduces the computational overhead at the receiver. However, it yields three drawbacks. First, the scheme only supports the AND gate in the access structure. Thus, it is less expressive comparing to the monotone structure proposed in the original CP-ABE design [37], which supports $k$-threshold gates. Secondly, the scheme uses the number of clouds, $m$, as a system-wide parameter, which makes the system structure very rigid. It requires all the receiver to use the same value for $m$, and imposes an additional burden to the sender, who is required to split the access structure into $m$ pieces and distribute to $m$ cloud servers. As individual users may have different needs regarding the protection of the attribute privacy, schemes that provide a flexible privacy setting are more desirable. Finally, the parallel-scheme incurs unnecessary communication between a cloud and the receiver. For example, when $Cloud_i$ successfully matches $\mathbb{A}_i$ to $S_i$ but $Cloud_j$ fails to match $\mathbb{A}_j$, $Cloud_i$ has to partially decrypt the message and forward the intermediate result to the receiver, regardless of the fact that the message cannot be decrypted by the user eventually. Frequent partial matchings incur an increasing computation and communication overhead to the clouds. To overcome these drawbacks, we propose an enhanced scheme using a chain-cloud structure and present it in the next section.

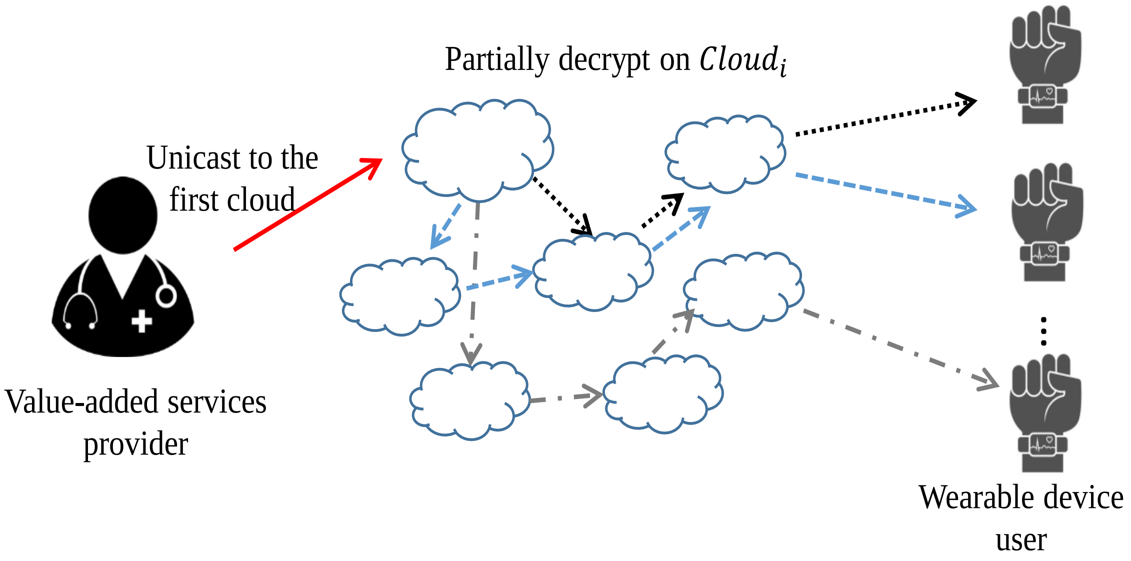

Partially decrypt on $Cloud_i$
Unicast to the first cloud
Value-added services provider
Wearable device user

Figure 7.2: Framework of privacy-preserving targeted broadcast in e-healthcare using chain-cloud scheme.

## 7.4 Chain-cloud Scheme

### 7.4.1 Overview and System Model

Unlike the parallel-cloud scheme, which requires the senders to connect to a same $m$ cloud servers simultaneously, the chain-cloud scheme allows each receiver to specify how many clouds to use and how attributes are delegated to each cloud. As shown in Figure 7.2, three receivers choose three sets of clouds to form three different paths to three devices. Moreover, the chain-cloud scheme allows the sender to encrypt a message under an expressive access policy $\mathbb{A}$ with $k$-threshold gates. That is, an access policy can be satisfied with any $k$ or more attributes.

To support this, we employ a Bloom filter [39] in each cloud to hold the part of attributes delegated to it. In particular, when a cloud receives an encrypted message, it first checks if the attributes that are delegated to it satisfy the access structure: if so, the cloud server partially decrypts the message and sends the result to the receiver; otherwise, it further looks up all the attributes in the access policy against the Bloom filter to check if the attributes have a chance to satisfy the access structure. If the Bloom filter look-up returns a positive result, it indicates that it is possible the receiver has the required attribute(s) to satisfy the access structure. So, the cloud should decrypt the attributes in the access structure as many as possible and forward the partially decrypted message to a next cloud. If the Bloom filter returns a negative result, which means the user does not have the required attribute(s) for sure, the cloud should stop propagating the message. Starting from the first cloud, this process repeats until either a cloud in the path finds that the access structure cannot be satisfied, or the partially decrypted message is successfully forwarded to the receiver.

### 7.4.2 Construction

**Setup($\lambda, N$).** The setup algorithm is similar to the one in the parallel-cloud scheme, except that it does not require a specific value for $m$, i.e., the number of clouds to be used.

**BloomFilterGen($S, [Cloud_i]$).** In the parallel-cloud scheme, the user (i.e., receiver) decides the clouds to be used and calls the BloomFilterGen algorithm, which takes the user's attribute set $S$ and the number of clouds $m$ as input to generate $m$ Bloom filters. Therefore, the user has a full control in deciding how to delegate her attributes to multiple clouds and in what order, by taking the sensitiveness of her attributes and her trust to cloud service providers into account. This not only provides a flexible mechanism for the user to determine her own multi-cloud platform setting, but also fits the real-world cloud usage scenario. That is, health-related attributes are likely to be stored in a private hospital cloud (and thus more trusted), while profile attributes can be delegated to public clouds.

The selected $m$ clouds are organized as a chain. Each cloud delegates a subset of attributes ($S_i$), from which the Bloom filter is generated. Here, we use an example to explain the generation process. Suppose the size of the universal attribute set $N$ is 200 and the user selects three clouds, where $Cloud_1$ is responsible for the attributes in $S_1 \subseteq [1,100]$, $Cloud_2$ for $S_2 \subseteq [101,160]$ and $Cloud_3$ for $S_3 \subseteq [161,200]$. To build the Bloom filter $BF_1$ for $Cloud_1$, the attributes $j \in S_2$ and $j' \in S_3$ are inserted into the $BF_1$. We can adjust the false positive rate by changing the size of the Bloom filter and the number of hash functions. To further increase the probability of false positives, we randomly set some bits in the filter to 1 to introduce noise. As shown in Figure 7.3,

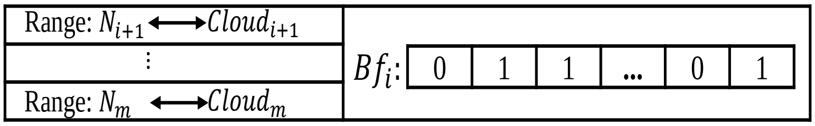


Figure 7.3: The Bloom filter sent to each cloud.

our Bloom filer consists of two parts. The first part is a lookup table used by a cloud to locate the next cloud, which is in charge of a specific range of attributes, and the second part is the noisy Bloom filter, which contains attributes in $Cloud_{i+1}, \cdots, Cloud_m$, and the noise bits.

In the chain-cloud scheme, each user has her own cloud usage specification ($CUS$) for attribute delegation and can change it anytime without informing the senders. This makes it more flexible than the parallel-cloud scheme. Then, the user sends the cloud usage specification to the TA to correctly distribute the corresponding transformation keys to the clouds in use.

**KeyGen($MK, S, t, CUS$).** The KeyGen algorithm takes as input the master key, a user's attribute set $S$ and private key $t$, and her cloud usage specification $CUS$. It generates the private key $SK$ and the transformation key $TK$ in the same way as described in Section 7.3.2. Then, the TA sends the corresponding transformation key $TK$ to each cloud according to the $CUS$.

**Encrypt($PK$, $M$, $\mathbb{A}$).** The chain-cloud scheme can support expressive access policies such as access tree defined in the original CP-ABE [37]. Here, we take access tree as an example to briefly explain the encryption algorithm.

Let $T$ be the access tree representing the access structure $\mathbb{A}$. Each non-leaf node $x$ of the tree is a threshold gate associated with a threshold value $k_x$, where $0 < k_x \leq num_x$ for a node with $num_x$ children. A leaf node is associated with an attribute and a threshold $k = 1$. We index the children of each node from 1 to $num_x$, and use $index(x)$ to return the index value of $x$ among its sibling nodes.

Upon receiving an encrypted message, the cloud checks if the access tree is satisfied. Let $T_x$ denote a subtree rooted at node $x$. If a set of attributes $\gamma$ satisfies the access tree $T_x$, we denote it as $T_x(\gamma) = 1$. $T_x(\gamma)$ can be computed recursively: for a leaf node $x$, $T_x(\gamma)$ returns 1 if the attribute of

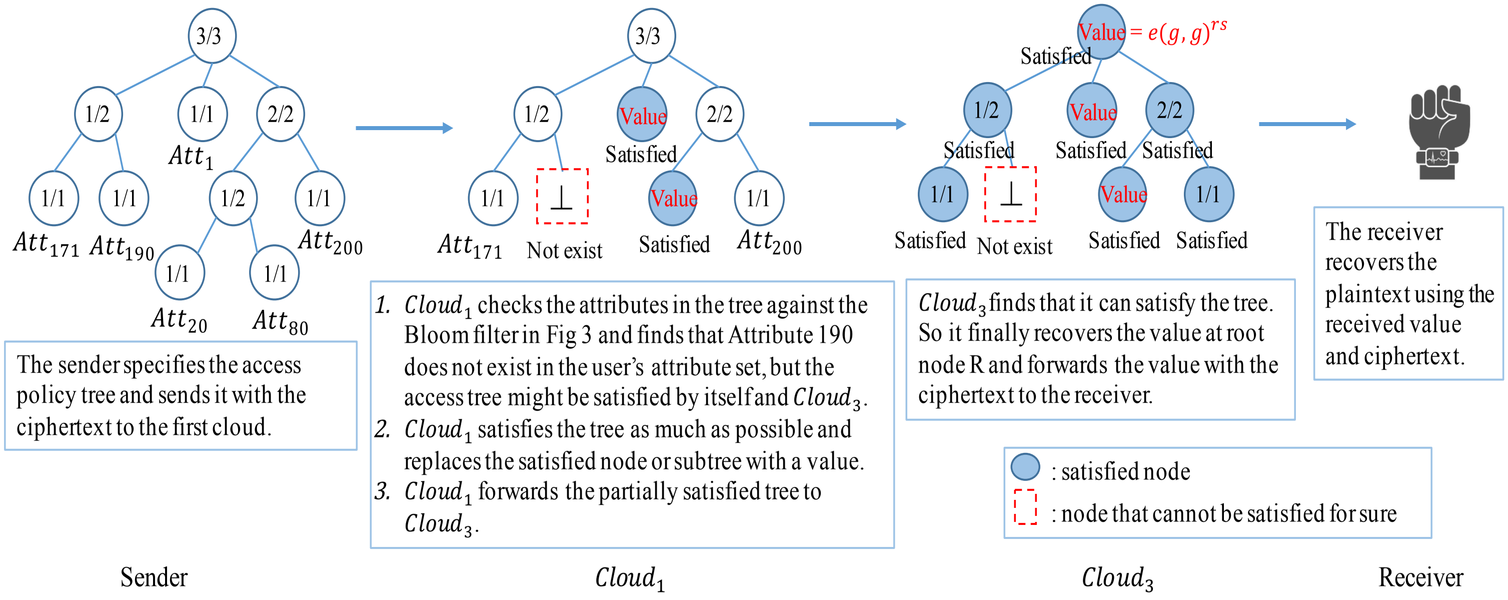

3/3
1/2
1/1
2/2
1/1
1/1
$Att_1$
1/2
1/1
$Att_{171}$
$Att_{190}$
$Att_{200}$
1/1
1/1
$Att_{20}$
$Att_{80}$
The sender specifies the access policy tree and sends it with the ciphertext to the first cloud.
Sender
3/3
1/2
Value
2/2
Satisfied
1/1
⊥
Value
1/1
$Att_{171}$
Not exist
Satisfied
$Att_{200}$
1. $Cloud_1$ checks the attributes in the tree against the Bloom filter in Fig 3 and finds that Attribute 190 does not exist in the user's attribute set, but the access tree might be satisfied by itself and $Cloud_3$.
2. $Cloud_1$ satisfies the tree as much as possible and replaces the satisfied node or subtree with a value.
3. $Cloud_1$ forwards the partially satisfied tree to $Cloud_3$.
$Cloud_1$
Value = $e(g,g)^{rs}$
Satisfied
1/2
Value
2/2
Satisfied
Satisfied
Satisfied
1/1
⊥
Value
1/1
Satisfied
Not exist
Satisfied
Satisfied
$Cloud_3$ finds that it can satisfy the tree. So it finally recovers the value at root node R and forwards the value with the ciphertext to the receiver.
● : satisfied node
⬚ : node that cannot be satisfied for sure
$Cloud_3$
The receiver recovers the plaintext using the received value and ciphertext.
Receiver

Figure 7.4: A transformation example: attributes set[1,100] is outsourced to $Cloud_1$, [101, 160] to $Cloud_2$, and [162,200] to $Cloud_3$.

$x$ is in $\gamma$; for a non-leaf node $y$, the cloud evaluates $T_{y'}(\gamma)$ for all the children $y\prime$ of node $y$, and sets $T_y(\gamma)$ to 1 if at least $k_y$ children return 1.

To encrypt a message, the Encrypt() algorithm first chooses a polynomial $q_x$ in a top-down manner, starting from the root $R$, for each node $x$ in the tree. For each node $x$ in the tree, the degree $d_x$ of the polynomial $q_x$ is set to $k_x - 1$, where $k_x$ is the threshold. For the root $R$, the algorithm chooses a random secret $s \in \mathbb{Z}_p$ and sets $q_R(0) = s$. Then, it chooses $d_R$ additional random values to construct $q_R$ completely. For any other node $x$, it sets $q_x(0) = q_{parent(x)}(index(x))$, where $parent(x)$ denote the parent node of $x$, and chooses $d_x$ additional randoms to define $q_x$.

Let $Y$ be the set of leaf nodes in $\mathbb{A}$. Once the access tree is defined, the algorithm encrypts the message as below and sends the the ciphertext to the first cloud server in the chain.

$$\begin{aligned} CT =& (\mathbb{A}, \widetilde{C} = Me(g,g)^{\alpha s}, C = h^s, \\ & \forall y \in Y : C_y = g^{q_y(0)}, C'_y = H(y)^{q_y(0)}). \end{aligned}$$

**Transform($CT, TK_i$).** When $Cloud_i$ receives the partially decrypted ciphertext from the previous cloud in the chain, it calls Transform() for further decryption.

For a leaf node $x$ with attribute $j \in N_i$ and $j \in S_i$, $Cloud_i$ calls $DecryptNode()$, which is described in Section 7.3.2, and computes $\frac{e(D_j, C_j)}{e(D'_j, C'_j)} = e(g,g)^{rq_j(0)}$. The node is then marked as $satisfied$ and the node value is set to $e(g,g)^{rq_j(0)}$.

If $j \in N_i$ but $j \notin S_i$, which means attribute $j$ is in charge of $Cloud_j$ but does not belong to the attribute set $S_j$ of the user, we should set the node value to an error symbol $\perp$, indicating $unsatisfied$.

If $j \notin S_i$ but $j \notin N_i$, the attribute $j$ (i.e., a leaf node) may be outsourced to another cloud. So, $Cloud_i$ looks it up in the Bloom filter and sets the node to $\perp$ if the result is negative. Note that, a positive result cannot guarantee the ownership of the attribute due to the false positive introduced by the Bloom filter. Also, not all the leaf nodes are associated with an attribute. Some are the internal nodes before the transformation done by the previous cloud.

We then compute the value for the non-leaf nodes in a bottom-up manner. For a non-leaf node $x$, we choose an arbitrary $k_x-$sized set $E_x$ of its child nodes $z$ to test if it satisfies node $x$. If such set exists, we compute the value of $x$ as $V_x$:

$$\begin{aligned}V_x &= \prod_{z \in E_x} V_z^{\Delta_{i,E'_x(0)}}, where\ i{=}index(z)\ and\ E'_x{=}\{index(z){:}z{\in}E_x\} \\ &= \prod_{z \in E_x} (e(g,g)^{rq_z(0)})^{\Delta_{i,E'_x}(0)} \\ &= \prod_{z \in E_x} (e(g,g)^{r \cdot q_{parent(z)}(index(z))})^{\Delta_{i,E'_x}(0)} \\ &= \prod_{z \in E_x} e(g,g)^{r \cdot q_x(i) \cdot \Delta_{i,E'_x}(0)} \\ &= e(g,g)^{r \cdot q_x(0)}.\end{aligned}$$

Otherwise, the value of $x$ is set to $\perp$. Repeatedly, we compute the value of the root node $R$ as $V_R = e(g,g,)^{rq_R(0)} = e(g,g)^{rs}$.

Next, $Cloud_i$ computes $\widetilde{D} = e(g,g,)^{st(\alpha+r)}$ as described in Section 7.3.2, and forwards $\widetilde{C}, \widetilde{D}, V_R$ to the receiver. If the computation stops before arriving at root $R$ but the Bloom filter indicates that an attribute satisfying the access structure may exist in subsequent cloud servers, $Cloud_i$ looks up the table in Figure 7.3 and forwards the intermediate result to the next cloud that probably hold the attributes to satisfy the entire tree. If no such cloud exists, it stops propagating and thus reduces the computation and communication overhead in the subsequent clouds. In this way, the noisy Bloom filter designed for the chain-cloud scheme increases the transformation efficiency. Only when an access policy can probably be satisfied by certain subsequent clouds, the current cloud forwards the message.

As shown in the example in Figure 7.4, when $Cloud_1$ finds that the access tree is satisfied by itself and $Cloud_3$, it skips $Cloud_2$ and directly forwards the intermediate result to $Cloud_3$.

**Decrypt($\widetilde{C}, \widetilde{D}, V_R, SK$).** Once receiving the partially decrypted message from a cloud server, the receiver performs the final decryption operation as below.

$$\begin{aligned}\frac{\widetilde{C}}{\frac{(\widetilde{D})^{1/t}}{V_R}} &= \frac{\widetilde{C}}{\frac{(e(g,g)^{st(\alpha+r)})^{1/t}}{e(g,g)^{rs}}} && = \frac{\widetilde{C}}{\frac{e(g,g)^{s(\alpha+r)}}{e(g,g)^{rs}}} \\ &= \frac{Me(g,g)^{\alpha s}}{e(g,g)^{\alpha s}} && = M.\end{aligned}$$

Obviously, the decryption overhead on resource-constrained devices is reduced to one exponentiation operation in the chain-cloud scheme.

**Verify($M, \sigma, V$).** The verification algorithm of the chain-cloud scheme is similar to the one in the parallel-cloud scheme discussed in Section 7.3.2. The sender counts the message encrypted under the access policy $\mathbb{A}$, generates the verification data $V = senderID|H_1(\mathbb{A})|seq(H_1(\mathbb{A}))|T$, and signs the message and verification data as $\sigma = sign(H_1(M|V))$.

Once the receiver recovers the plaintext $M$ and the verification value $V$, she first checks them against the signature $\sigma$ to verify the correctness of the transformation, and then verifies the completeness by comparing if the received sequence number is greater than the stored sequence number by 1, for the access policy $\mathbb{A}$.

## 7.5 Security Analysis and Performance Evaluation

In this section, we first analyze the security features of the proposed multi-cloud ABE schemes, and then explain our implementation of the two schemes on Amazon EC2 and Microsoft Azure.

### 7.5.1 Security Analysis

**Data Privacy.** The sender does not want the clouds or other unauthorized parties to access the message it sends to a target group of users. From the decryption algorithms in Section 7.3.2 and Section 7.4.2, we see that an adversary needs to be able to cancel out $e(g,g)^{\alpha s}$ from the ciphertext $\widetilde{C}$ to recover the plaintext message. In doing so, he needs to be able to compute the pairing value over $C$ from the ciphertext and $D$ from the transformation key, respectively, to cancel out the secret $e(g,g)^{rs}$. The security of the pairing operation ensures that an unauthorized party without knowing the correct $r$ and $s$ cannot recover this secret. Meanwhile, since the paring value is blinded by the private key $t$ in our scheme, it is impossible for a cloud to recover the plaintext, even though it has access to $e(g,g)^{rs}$.

**Attribute Privacy.** To reduce the computational cost at the devices, the operation of matching the attributes to the access structure is outsourced to the cloud. As the attributes of the user are disclosed to the cloud, it introduces a serious privacy concern, especially when the attributes contain sensitive information about the user. One may argue that in real-world, attributes delegated to the clouds are represented in the form of a hash value, instead of the meaningful raw text, and thus incurs less privacy risk. Actually, a malicious cloud can still launch the dictionary attack to check every possible word against the hash value. Therefore, the attribute privacy is considered unprotected in all the existing outsourced ABE schemes using a single cloud.

In our multi-cloud schemes, each cloud server is only in charge of a part of attributes, so that no single cloud can learn the complete set of attributes of a user. This significantly reduces the privacy leakage caused by the attribute-based inference attacks due to outsourced decryption. To formally assess the improved protection to attribute privacy in our multi-cloud schemes, we define *accuracy* as a measure of the degree of knowledge about a user, and compare with the outsourced ABE schemes using a single cloud. The higher the accuracy, the more the cloud knows about a user.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

where *TP, FP, TN, FN* represent true positive, false positive, true negative, and false negative, respectively. Specifically, *TP* denotes the number of attributes that the cloud is certain that a user has, and *TN* denotes the number of attributes that the cloud is certain that a user does not have. From the definition, we see that the lower bound of accuracy is 0.5, when the cloud has no knowledge about a user and thus can only guess with a probability of 0.5.

Assume the universal attribute set is $N$, a user's attribute set is $S$, and the number of clouds is $m$. In the single-cloud outsourced ABE, the accuracy achieves its upper bound, $(|S|+|N-S|)/|N|=1$, since the cloud clearly knows that the user owns $|S|$ attributes and does not own the remaining $|N-S|$ attributes.

In the parallel-cloud scheme, each cloud server works independently from other clouds. It knows only the attributes outsourced to itself, i.e., $|S|/m$ attributes. For the remaining attributes, it can only guess with a probability of 0.5. So, the accuracy is at most $(|S|/m+0.5(|N|-|S|/m))/|N|$. In a setting that $|N|=200, |S|=100, m=5$, the accuracy is 0.55. Obviously, the parallel-cloud scheme improves attribute privacy significantly compared to the single-cloud outsourced ABE.

In the chain-cloud scheme, for the sake of efficiency, each cloud stores a Bloom filter to check if the access policy has a chance of being satisfied by the subsequent clouds. This incurs attribute privacy leakage. However, the degree of leakage is controllable by the user by carefully selecting the cloud and determining its position in the cloud chain. This is because the cloud which is closer to the tail of the chain cannot test the membership of any attribute in the clouds that are closer to the head of the chain. Moreover, the user can further adjust the false positives caused by the Bloom filter by adding noise, changing its size and the number of hash functions. In an extreme case, to prevent a cloud from inferring the attributes in other clouds from the Bloom filter, all positions in a Bloom filter need to be set to 1. With these noise bits, the cloud forwards all the messages that are not satisfied so far, which is actually equivalent to disabling the Bloom filter. As a result, each cloud knows only the attributes it holds.

To measure the accuracy, we need to set $p_{FP}$, which is the probability of false positive of the Bloom filter, and compute:

$$\frac{\frac{|S|}{m}+|N-S|*(1-p_{FP})+0.5(|N|-\frac{|S|}{m}-|N-S|*(1-p_{FP}))}{|N|}$$

where the fist part of the numerator is the true positives of attributes that are outsourced to the cloud, the second part is the true negatives that the cloud gets from the Bloom filter, and the third part is the correct guess of the remaining attributes with the probability of 0.5.

Let us set $p_{FP}$ to 0.6, for example, the accuracy on the fist cloud of the chain, who knows the most information about a user, is 0.65. We see that the cloud gets more accurate information about a user from the chain-cloud scheme than the parallel-cloud scheme. When the probability of false positive is set to 1, it becomes equal to that in the parallel-cloud scheme.

**Access Policy Privacy.** In the single-cloud outsourced ABE, a cloud server can see the complete access policy and further infer the underlying message. In our parallel-cloud scheme, since an access policy is divided into multiple pieces, each cloud knows only a part of the policy. In the chain-cloud scheme, to support the flexible system structure and the complete expressiveness of the access policy, the access structure is distributed over the chain. Any cloud in the chain has no knowledge about the attributes in previous clouds, since the attributes that are satisfied in previous clouds have been replaced with node values, which looks like a random value. Therefore, a cloud that is closer to the tail sees less about access structure. Besides, the chain is organized in a way that more trusted clouds are placed at the beginning positions, thus it is reasonable to assume that allowing them to see a more access structure will not cause severe privacy leakage.

**Collusion Resistance.** A major challenge to construct a secure ABE scheme is to prevent colluding users so that they cannot combine their attributes to satisfy an access policy, which they cannot decrypt individually. Our proposed schemes are resistant to the colluding attacks. Similar as the original CP-ABE design, we select a random $r$ for each user in the key generation algorithm, which results in distinct values for different users when recovered by the secret sharing scheme.

**Verifiability.** Some messages from the sender may include critical content, such as control commands, and thus the correctness verification of the transformation is very important. We use the public-key signature scheme in both schemes to enable end-to-end verification. Since the private key for singing the message is only known to the sender, no cloud nor adversary can forge a valid signature.

Another challenge is to verify the completeness of the transformation. A cloud may accidentally fail in matching an access policy to the attributes, due to system errors or intentional misbehavior. To verify the completeness, the sender and the receiver need to share a common knowledge

| Scheme | Complexity of decryption |
|---|---|
| CP-ABE | (2n+1)P + 2$\mathbf{M}_2$ |
| Parallel-cloud CP-ABE | $\mathbf{E}_2$ + 2$\mathbf{M}_2$ |
| Chain-cloud CP-ABE | $\mathbf{E}_2$ + (m+2)$\mathbf{M}_2$ |

Table 7.1: Comparison of asymptotic complexity of decryption operation of different scheme.

about how many messages are transmitted. We adopt a stateful verification scheme for completeness, which maintains a continuously increasing counter for the messages that are encrypted under a specific access policy $\mathbb{A}$ as the shared knowledge. Since the counter is also signed by the sender, the receiver can trust it to verify if any message is accidentally or maliciously discarded. However, this completeness verification still has limitations. If a malicious cloud never forwards a message under a specific access structure, which should have been satisfied by the receiver, the receiver cannot build the shared knowledge and know the existence of a message without interacting with the sender. A simple yet effective countermeasure for the parallel-cloud scheme is that the sender sends the complete access policy $\mathbb{A}$ instead of a part $\mathbb{A}_i$ to each cloud and the cloud forwards it to the receiver. The parallel-cloud scheme introduces redundancy for completeness verification. As long as at least one cloud is honest, the receiver can verify if she matches the access policy, and determines if a malicious cloud exists. The receiver can afford such lightweight matching operation as it only needs to compare if two attributes are identical. However, the drawback of this countermeasure is that the cloud knows the complete access policy, which may cause privacy leakage. We argue that this is a reasonable price to pay, considering the criticalness of the completeness. Moreover, completeness verification remains a challenging task for the chain-cloud scheme. Since only the last cloud successfully satisfying the access policy will forward the message to the receiver, it does not provide redundancy as the parallel-cloud scheme does. In fact, the completeness verification is still a challenging task even for the general outsourcing applications such as searchable encryption, which involves only one cloud. To the best of our knowledge, only accumulator [123] can provide the completeness verification at a very high cost, and there is no known solution for outsourced ABE scheme. We consider completeness verification for the chain-cloud outsourced ABE scheme an open problem for our future work.

|  | CP-ABE | OP-CP-ABE | OC-CP-ABE |
|---|---|---|---|
| 2 clouds | 1444 | 830 | 1620 |
| 5 clouds | 1444 | 732 | 1931 |

Table 7.2: Comparison of the delay between the sending time and the receiving time in the 2-cloud and 5-cloud settings with 60 attributes in the access policy.

### 7.5.2 Performance Evaluation

We implement the proposed parallel-cloud and chain-cloud outsourced ABE schemes in real-world clouds, i.e., Amazon EC2 and Microsoft Azure, and compare the performance of our schemes with the one of the original CP-ABE [37] in terms of asymptotic complexity and the experimental performance.

Since we do not make changes to the encryption algorithm, and the partial decryption is delegated to the cloud which is assumed to have unlimited computation capability, we focus on the comparison of the overhead at the recipient devices. Note that in the implementation, a message itself is actually encrypted using AES keys, which has fixed computational overhead. So, we only evaluate the overhead introduced by the ABE operations.

Table 7.1 compares the asymptotic complexity of the three schemes, where **P** denotes the paring operation, $\mathbf{E_2}$ denotes the group exponentiation, $\mathbf{M_2}$ denotes the group multiplication in $\mathbb{G}_2$, $n$ denotes the number of attributes in the access policy, and $m$ denotes the number of clouds used in the parallel-cloud outsourced ABE. Obviously, the computational complexity of our schemes is independent from the complexity of access policy, which only needs a constant time to recover the plaintext.

Next, we implement the original CP-ABE and our schem- es using the Java Pairing-Based Cryptography Library [13]. We use Type A elliptic curve of 160-bit group order, which provides 1024-bit discrete log security equivalently. The experiments are conducted on the Raspberry Pi 2 [18], with 700MHz ARM A6 microprocessor and 512 MB RAM, to simulate the resource-constrained IoT devices such as smart phones acting as the gateway for wearable devices and smart meters. We launch multiple Amazon EC2 and Windows Azure instances to simulate the cloud

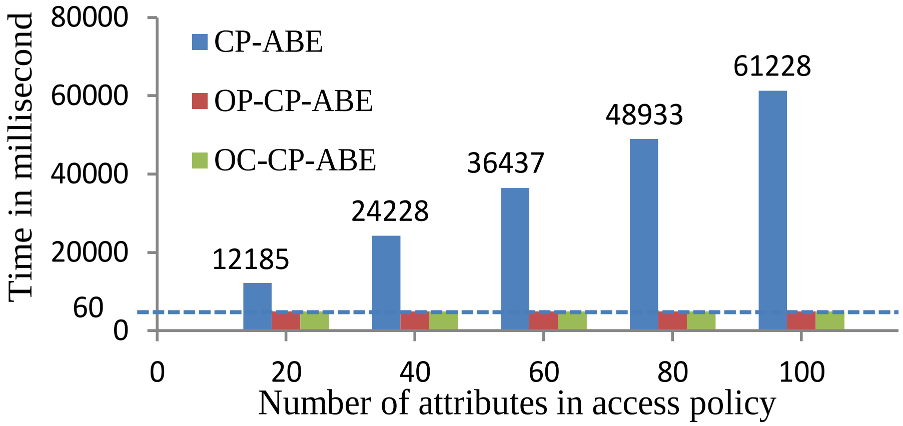


Figure 7.5: Comparison of the decryption time between the original CP-ABE, the parallel-cloud outsourced CP-ABE and the chain-cloud outsourced CP-ABE with varying number of attributes in the access structure.

service providers in our schemes. The partial decryption in the clouds in the parallel-cloud scheme is obviously efficient since all the cloud servers decrypt the corresponding pieces simultaneously. However, for the chain-cloud scheme, since the partial decryption is conducted in a sequential manner, we are interested in evaluating the delay introduced by the chain structure.

Figure 7.5 compares the decryption time of the original CP-ABE, our proposed parallel-cloud CP-ABE and chain-cloud CP-ABE on the Raspberry Pi 2, using a multi-cloud platform of 5 clouds. Our asymptotic complexity analysis is confirmed by the real implementation, that is, the decryption time of the original CP-ABE is proportional to the complexity of the access structure, while the overhead in our schemes is significantly reduced to a constant value, regardless of the access structure. This shows the benefit of lightweight decryption introduced by the outsourcing.

Finally, we evaluate the delay introduced by the multi-cloud structure. To evaluate the chain-cloud scheme, we connect the instances of EC2 and Azure alternately to measure the delay, since the communication time between two servers from the same cloud service provider (e.g., two Amazon instances) is negligible. The round trip time (RTT) of two EC2 virtual machines (VMs) is less than 2 milliseconds, while the RTT between an EC2 VM and an Azure VM is around 60 milliseconds. The sender and the recipient are synchronized through a Socket communication.

Table 7.2 compares the delay between the time of sending out the ciphertext and the time of receiving all the partially decrypted ciphertexts from the cloud servers. In this experiment, the message is encrypted under an access policy with 60 attributes. We use the single-cloud outsourced ABE as the base line, and compares the delay in the 2-cloud and 5-cloud settings. We see that the parallel-cloud scheme achieves the best performance in terms of delay, since each cloud partially decrypts a small part of the ciphertext and transmits it in a parallel way. Compared to the base line, the chain-scheme has a larger delay, because all the clouds need to sequentially decrypt the ciphertext, which introduces the transmission delay and the delay caused by the serialization and un-serialization of data for network transmission.

## 7.6 Related Work

**Attribute-based messaging.** Extensive studies have been done on secure data publishing. Bobba et al. developed an attribute-based messaging system where senders can dynamically create a list of recipients based on their attributes [40]. However, this scheme incurs high computational overhead and thus only suits for traditional PC-based applications. Fadlullah et al. proposed a secure targeted broadcast scheme for smart grid where the utility encrypts a message using key policy attribute-based encryption (KP-ABE) and broadcasts the ciphertext to a specific group of users [65]. KP-ABE is the first ABE scheme introduced by Sahai and Waters [78]. In KP-ABE, each encrypted message is labeled with a set of attributes and each user is assigned a private key associated with an access structure. A user decrypts an encrypted message only when the attributes associated with the ciphertext satisfy her access policy. Compared to CP-ABE, KP-ABE is less expressive in specifying who has access to the encrypted message. Fadlullah's ABE-based broadcast exploits the original KP-ABE construction without outsourcing the decryption operation. Although it avoids the problem of attribute privacy leakage, it imposes a very large computational overhead on receivers, which is prohibitively high for resource-constrained devices. In [87], the authors proposed a practical ABE-based data sharing scheme using CP-ABE and the outsourcing technique to reduce the overhead on smart meters. However, this scheme is still susceptible to the privacy leakage risk, that is, when a user delegates all the attributes to a single cloud, the cloud is able to infer her sensitive information from the attributes.

**Verifiable outsourced attribute-based encryption.** The presented work is also related to verifiable outsourced ABE. While the outsourced schemes [79, 167, 87] protect the data privacy, they provide no guarantee to the correctness of the transformation performed by the cloud server. Lai et al. implemented a verifiable outsourced ABE approach by attaching an additional encrypted random message to the real message and computing the digest of the two messages together [99]. The receiver recovers both messages and checks the digest to verify the correctness of the received messages. Similarly, Li et al. proposed an outsourced ABE with checkability by adding a redundant pre-shared k-length bit string to each message. The receiver can detect the dishonest action by checking the bit string. Although these schemes can verify the correctness of the transformation at the cloud server, they provide no guarantee to the completeness of the forwarded messages. That is, checking whether the cloud completely forwards all messages that a user is qualified to receive. This problem is somehow related to verifiable computation [73, 123] and verifiable searchable encryption [165], which guarantee the returned result is correct and complete. However, these schemes rely on either expensive fully homomorphic encryption or the pre-sharing of certain knowledge about the underlying message between the sender and the receiver, which is obviously infeasible in the privacy-preserving targeted broadcast applications.

## 7.7 Summary

In this chapter, we present two multi-cloud outsourced-ABE schemes for privacy preserving targeted broadcast for IoT devices. By enforcing the collaboration between multiple clouds, our schemes significantly reduce the computational overhead at the resource-constrained IoT devices. The new schemes protect data privacy, attribute privacy and access policy privacy. To the best of our knowledge, this is the first work to utilize multi-cloud structure to prevent the disclosure of attributes and access policies in outsourcing. Our schemes also provide verifiability, which allows receivers to verify the correctness and completeness of the outsourced operations. Through intensive security analysis and experiments with simulated IoT devices and commercial cloud platforms, we demonstrate the security guarantees and outstanding performance of the proposed schemes.

# Chapter 8

# Conclusions

Internet of Things are extending to every facet of our lives, and thus, it is very critical to move fast to address these rising security and privacy concerns in IoT systems before severe disasters happen. In this dissertation, we mainly address the challenges in two domains: (1) how to protect IoT devices against cyberattacks using Tor; (1) how to leverage cloud-assisted techniques to protect sensitive data during storage, dissemination and utilization for IoT applications.

First, we propose two novel Tor hidden services based solutions to secure smart home: IoT gateway over multipath Tor hidden services and IoT gateway with separate command and data channels. In the first solution, we use multipath routing scheme to improve the performance of Tor hidden services. With all data routed in the Tor network, this solution suits users who desire strong security but acceptable performance. To further improve the performance, our second solution transfers small-size command through Tor and transfers bulk data through the Internet so that we can leverage both the strong security of Tor and the good performance of the Internet.

Second, for those smart home users who require very strong security but care less about the performance, we enhance the resistance of Tor hidden service to traffic analysis attacks. We propose a different multipath routing based scheme that exploits flow mixing and flow merging to distort or destroy inserted traffic patterns in a victim's flow.

Another important contribution of this dissertation is to explore the potentials of cloud-assisted

IoT security. We present a reliable, searchable and privacy-preserving e-healthcare system, which enables users to store private data on the cloud in an encrypted format, and meanwhile health service providers can search over encrypted data.

Our fourth contribution is to apply machine learning classification over encrypted IoT data stored on the cloud. We propose a cloud-assisted, privacy-preserving machine learning classification scheme over encrypted data for IoT devices.

At last, we pay attention to the privacy-preserving data publishing scheme for IoT applications. We explore the problem of privacy-preserving targeted broadcast in IoT, and propose two multi-cloud-based outsourced-ABE (attribute-based encryption) schemes, namely the parallel-cloud ABE and the chain-cloud ABE.

# References

[1] 9 baby monitors wide open to hacks that expose users' most private moments. https://arstechnica.com/security/2015/09/9-baby-monitors-wide-open-to-hacks-that-expose-users-most-private-moments/.

[2] A Definition of the Internet of Things (IoT). http://iot.ieee.org/definition.html.

[3] Anonymizer.com. https://www.anonymizer.com/.

[4] ARK-1123H: Intel IoT Gateway. https://solutionsdirectory.intel.com/solutions-directory/ark-1123h-ultra-small-and-ark-2121l-multiple-io-advantech-iot-gateway-starter.

[5] Censys. https://censys.io/.

[6] DeepMind wants its healthcare AI to charge by results — but first it needs your data. https://techcrunch.com/2016/09/20/deepmind-wants-its-healthcare-ai-to-charge-by-results-but-first-it-needs-your-data/.

[7] Hack Samsung Fridge. https://www.pentestpartners.com/security-blog/hacking-defcon-23s-iot-village-samsung-fridge/.

[8] Hackers Make the First-Ever Ransomware for Smart Thermostats. https://motherboard.vice.com/en_us/article/aekj9j/internet-of-things-ransomware-smart-thermostat.

[9] Hacking 14 IoT Devices. https://www.iotvillage.org/slides_DC23/IoT11-slides.pdf.

[10] Home Assistant. https://home-assistant.io/.

[11] Intel IoT Gateway. https://www.intel.com/content/www/us/en/internet-of-things/gateway-solutions.html.

[12] Internet of Things Global Standards Initiative. http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx.

[13] Java Pairing-Based Cryptography Library. http://gas.dia.unisa.it/projects/jpbc/.

[14] Let's Encrypt. https://letsencrypt.org/.

[15] libevent. http://libevent.org/.

[16] Openhab. https://www.openhab.org/.

[17] Ransomware Ruins Holiday By Hijacking Family's LG Smart TV on Christmas Day. https://www.yahoo.com/tech/ransomware-ruins-holiday-hijacking-familys-201136667.html.

[18] Raspberry Pi. https://www.raspberrypi.org/.

[19] Shodan. https://www.shodan.io/.

[20] Smartthings. http://www.samsung.com/us/smart-home/smartthings/hubs/f-hub-us-2-f-hub-us-2/.

[21] The 10 most popular Internet of Things applications. http://iot-analytics.com/10-internet-of-things-applications/.

[22] VStarCam Eye4. http://www.eye4.so/.

[23] Trendnet cameras - i always feel like somebody's watching me. `http://console-cowboys.blogspot.com/2012/01/trendnet-cameras-i-always-feel-like.html`, 2012.

[24] Hacking defcon 23's iot village samsung fridge. `https://www.pentestpartners.com/security-blog/hacking-defcon-23s-iot-village-samsung-fridge/`, 2015.

[25] KasperskyOS. `https://os.kaspersky.com/wp-content/uploads/sites/11/2017/02/KasperskyOS-for-IoT-En.pdf`, 2017.

[26] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *ACM Sigmod Record*, volume 29, pages 439–450. ACM, 2000.

[27] Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. LASTor: A Low-Latency AS-Aware Tor Client. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, May 2012.

[28] Mashael Alsabah, Kevin Bauer, Tariq Elahi, and Ian Goldberg. The path less travelled: Overcoming tor's bottlenecks with traffic splitting. In *Proc. of the 13th Privacy Enhancing Technologies Symposium*, July 2013.

[29] Mashael AlSabah, Kevin Bauer, and Ian Goldberg. Enhancing tor's performance using real-time traffic classification. In *Proceedings of the 19th ACM conference on Computer and Communications Security*, October 2012.

[30] Mashael AlSabah, Kevin Bauer, Ian Goldberg, Dirk Grunwald, Damon McCoy, Stefan Savage, and Geoffrey Voelker. Defenestrator: Throwing out windows in tor. In *Privacy Enhancing Technologies*, pages 134–154. Springer, 2011.

[31] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):1–30, 2006.

[32] Giuseppe Ateniese, Luigi V Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *International Journal of Security and Networks*, 10(3):137–150, 2015.

[33] Dylan Ayrey. Why are jtag and uart still effective attack vectors for iot devices? `https://p16.praetorian.com/blog/why-are-jtag-and-uart-still-effective-attack-vectors-for-iot-devices`.

[34] CJ Barker. Mirai (DDoS) Source Code Review. `https://medium.com/@cjbarker/mirai-ddos-source-code-review-57269c4a68f`.

[35] Sebastien Barre, Christoph Paasch, and Olivier Bonaventure. Multipath tcp: from theory to practice. In *NETWORKING 2011*, pages 444–457. Springer, 2011.

[36] Elisa Bertino and Nayeem Islam. Botnets and internet of things security. *Computer*, 50(2):76–79, 2017.

[37] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Security and Privacy, IEEE Symposium on*, 2007.

[38] Alex Biryukov, Ivan Pustogarov, and R Weinmann. Trawling for tor hidden services: Detection, measurement, deanonymization. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 80–94. IEEE, 2013.

[39] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[40] Rakesh Bobba, Omid Fatemieh, Fariba Khan, Carl A Gunter, and Himanshu Khurana. Using attribute-based access control to enable attribute-based messaging. In *ACSAC*, 2006.

[41] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Proc. of EUROCRYPT'04*, 2004.

[42] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Annual International Cryptology Conference*, pages 258–275. Springer, 2005.

[43] Joppe W Bos, Kristin Lauter, and Michael Naehrig. Private predictive analysis on encrypted medical data. *Journal of biomedical informatics*, 50:234–243, 2014.

[44] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*, 2015.

[45] Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 37–54. Springer, 2003.

[46] Samuel Burke. Massive cyberattack turned ordinary devices into weapons. `http://money.cnn.com/2016/10/22/technology/cyberattack-dyn-ddos/index.html`, 2016.

[47] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Proceedings of NDSS'14*, 2014.

[48] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Proceedings of CRYPTO'13*. Springer, 2013.

[49] Patrik Cerwall. Ericsson mobility report. `http://www.ericsson.com/res/docs/2015/mobility-report/ericsson-mobility-report-nov-2015.pdf`, 2015.

[50] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Advances in Neural Information Processing Systems*, pages 289–296, 2009.

[51] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), February 1981.

[52] Rong Cheng, Jingbo Yan, Chaowen Guan, Fangguo Zhang, and Kui Ren. Verifiable searchable symmetric encryption from indistinguishability obfuscation. In *Proceedings of CCS'15*, pages 621–626. ACM, 2015.

[53] Devin Coldewey. Smart locks yield to simple hacker tricks. `https://techcrunch.com/2016/08/08/smart-locks-yield-to-simple-hacker-tricks/`, 2016.

[54] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, 2015.

[55] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 45–64. Springer, 2002.

[56] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of CCS'06*, pages 79–88. ACM, 2006.

[57] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions. In *Australasian Conference on Information Security and Privacy*, pages 416–430. Springer, 2007.

[58] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 2–15, May 2003.

[59] Ben Dickson. Why iot security is so critical. `https://techcrunch.com/2015/10/24/why-iot-security-is-so-critical/`.

[60] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[61] Wenliang Du and Zhijun Zhan. Using randomized response techniques for privacy-preserving data mining. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 505–510. ACM, 2003.

[62] Tariq Elahi, Kevin Bauer, Mashael AlSabah, Roger Dingledine, and Ian Goldberg. Changing of the guards: A framework for understanding and improving entry guard selection in tor. In *Proceedings of the 2012 ACM workshop on Privacy in the electronic society*, pages 43–54. ACM, 2012.

[63] Fatih Emekçi, Ozgur D Sahin, Divyakant Agrawal, and Amr El Abbadi. Privacy preserving decision tree learning over multiple parties. *Data & Knowledge Engineering*, 63(2):348–361, 2007.

[64] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 235–253. Springer, 2009.

[65] Zubair Md Fadlullah, Nei Kato, Rongxing Lu, Xuemin Shen, and Yousuke Nozaki. Toward secure targeted broadcast in smart grid. *Communications Magazine, IEEE*, 2012.

[66] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security Analysis of Emerging Smart Home Applications. In *Proceedings of the 37th IEEE Symposium on Security and Privacy*, 2016.

[67] Amos Fiat and Moni Naor. Broadcast encryption. In *Advances in Cryptology*, 1993.

[68] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM*

*SIGSAC Conference on Computer and Communications Security*, pages 1322–1333. ACM, 2015.

[69] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, November 2002.

[70] Nathan Freitas. Internet of onion things. `https://www.youtube.com/watch?v=j2yT-0rmgDA`, 2016.

[71] John Fruehe. The internet of things is about data, not things. `https://www.forbes.com/sites/moorinsights/2015/07/30/the-internet-of-things-is-about-data-not-things/#27cfb92927cf`.

[72] Gartner Inc. Gartner iot forecast. `http://www.gartner.com/newsroom/id/3598917`, 2017.

[73] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology*. 2010.

[74] Tuan Nguyen Gia, Igor Tcarenko, Victor K Sarker, Amir M Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. Iot-based fall detection system with energy efficient sensor nodes. In *Nordic Circuits and Systems Conference (NORCAS), 2016 IEEE*, pages 1–6. IEEE, 2016.

[75] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[76] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding Routing Information. In R. Anderson, editor, *Proceedings of Information Hiding: First International Workshop*, pages 137–150. Springer-Verlag, LNCS 1174, May 1996.

[77] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[78] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS*, 2006.

[79] Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of abe ciphertexts. In *USENIX Security Symposium*, volume 2011, 2011.

[80] Vincenzo Gulisano, Magnus Almgren, and Marina Papatriantafilou. Online and scalable data validation in advanced metering infrastructures. In *Innovative Smart Grid Technologies Conference Europe*. IEEE, 2014.

[81] Florian Hahn and Florian Kerschbaum. Searchable encryption with secure and efficient updates. In *Proceedings of CCS'14*. ACM, 2014.

[82] Matt Hamblen. As smartwatches gain traction, personal data privacy worries mount. `http://www.computerworld.com/article/2925311/wearables/as-smartwatches-gain-traction-personal-data-privacy-worries-mount.html`.

[83] M Shamim Hossain and Ghulam Muhammad. Cloud-assisted industrial internet of things (iiot)–enabled framework for health monitoring. *Computer Networks*, 101:192–202, 2016.

[84] Amir Houmansadr and Nikita Borisov. Swirl: A scalable watermark to detect correlated network flows. In *Proceedings of the Network and Distributed Security Symposium - NDSS'11*. Internet Society, February 2011.

[85] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. Rainbow: A robust and invisible non-blind watermark for network flows. In *Proceedings of the Network and Distributed Security Symposium - NDSS'09*. Internet Society, February 2009.

[86] Qinlong Huang, Licheng Wang, and Yixian Yang. Decent: Secure and fine-grained data

access control with policy updating for constrained iot devices. *World Wide Web*, pages 1–17, 2017.

[87] Junbeom Hur. Attribute-based secure data sharing with hidden policies in smart grid. *Parallel and Distributed Systems, IEEE Transactions on*, 2013.

[88] Rob Jansen and Nicholas Hopper. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *Proceedings of the Network and Distributed System Security Symposium - NDSS'12*, February 2012.

[89] Rob Jansen, Nicholas Hopper, and Yongdae Kim. Recruiting new Tor relays with BRAIDS. In *Proceedings of the 2010 ACM Conference on Computer and Communications Security (CCS 2010)*, October 2010.

[90] Rob Jansen, Aaron Johnson, and Paul Syverson. LIRA: Lightweight Incentivized Routing for Anonymity. In *Proceedings of the Network and Distributed System Security Symposium - NDSS'13*, February 2013.

[91] Rob Jansen, Paul Syverson, and Nicholas Hopper. Throttling Tor Bandwidth Parasites. In *Proceedings of the 21st USENIX Security Symposium*, August 2012.

[92] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on tor by realistic adversaries. In *Proceedings of the 20th ACM conference on Computer and Communications Security*, 2013.

[93] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Proceedings of FC'13*. Springer, 2013.

[94] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proceedings of CCS'12*. ACM, 2012.

[95] Jonathan Katz and Andrew Y Lindell. Aggregate message authentication codes. In *Proceedings of CT-RSA'08*, pages 155–169. Springer, 2008.

[96] Negar Kiyavash, Amir Houmansadr, and Nikita Borisov. Multi-flow attacks against network flow watermarking schemes. In *USENIX Security Symposium*, 2008.

[97] Sarah Kuranda. The 10 biggest data breaches of 2015 (so far). `http://www.crn.com/slide-shows/security/300077563/the-10-biggest-data-breaches-of-2015-so-far.htm`, 2015.

[98] Kaoru Kurosawa and Yasuhiro Ohtaki. How to update documents verifiably in searchable symmetric encryption. In *Proceedings of CANS'13*, pages 309–328. Springer, 2013.

[99] Junzuo Lai, Robert H Deng, Chaowen Guan, and Jian Weng. Attribute-based encryption with verifiable outsourced decryption. *Information Forensics and Security, IEEE Transactions on*, 8(8):1343–1354, 2013.

[100] John Leyden. Samsung smart fridge leaves gmail logins open to attack. `https://www.theregister.co.uk/2015/08/24/smart_fridge_security_fubar/`.

[101] Fengjun Li and Bo Luo. Preserving data integrity for smart grid data aggregation. In *SmartGridComm*, 2012.

[102] Fengjun Li, Bo Luo, and Peng Liu. Secure information aggregation for smart grids using homomorphic encryption. In *SmartGridComm*. IEEE, 2010.

[103] Jin Li, Xinyi Huang, Jingwei Li, Xiaofeng Chen, and Yang Xiang. Securely outsourcing attribute-based encryption with checkability. *Parallel and Distributed Systems, IEEE Transactions on*, 25(8):2201–2210, 2014.

[104] Ming Li, Shucheng Yu, Ning Cao, and Wenjing Lou. Authorized private keyword search over encrypted personal health records in cloud computing. In *Proceedings of ICDCS'11*, pages 383–392. IEEE, 2011.

[105] Yi-Pin Liao and Chih-Ming Hsiao. A secure ecc-based rfid authentication scheme integrated with id-verifier transfer protocol. *Ad Hoc Networks*, 18:133–146, 2014.

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

[107] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Annual International Cryptology Conference*, pages 36–54. Springer, 2000.

[108] Zhen Ling, Junzhou Luo, Yiling Xu, Chao Gao, Kui Wu, and Xinwen Fu. Security vulnerabilities of internet of things: A case study of the smart plug system. *IEEE Internet of Things Journal*, 2017.

[109] Zhen Ling, Junzhou Luo, Wei Yu, Xinwen Fu, Dong Xuan, and Weijia Jia. A new cell counter based attack against tor. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 578–589. ACM, 2009.

[110] Lanchao Liu, Mohammad Esmalifalak, Qifeng Ding, Valentine A Emesih, and Zhu Han. Detecting false data injection attacks on power grid by sparse optimization. *Smart Grid, IEEE Transactions on*, 5(2):612–621, 2014.

[111] Rongxing Lu, Xiaohui Liang, Xu Li, Xiaodong Lin, and Xuemin Sherman Shen. Eppa: An efficient and privacy-preserving aggregation scheme for secure smart grid communications. *Parallel and Distributed Systems, IEEE Transactions on*, 2012.

[112] Rongxing Lu, Xiaodong Lin, and Xuemin Shen. Spoc: A secure and privacy-preserving opportunistic computing framework for mobile-healthcare emergency. *Parallel and Distributed Systems, IEEE Transactions on*, 2013.

[113] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining light in dark places: Understanding the Tor network. In *Proc. of the 8th International Symposium on Privacy Enhancing Technologies*, 2008.

[114] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of

things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[115] Amir Modarresi and James PG Sterbenz. Multilevel iot model for smart cities resilience. In *Proceedings of the 12th International Conference on Future Internet Technologies*, page 7. ACM, 2017.

[116] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster Protocol — Version 2. IETF Internet Draft, July 2003.

[117] W. Brad Moore, Chris Wacek, and Micah Sherr. Exploring the potential benefits of expanded rate limiting in tor: Slow and steady wins the race with tortoise. In *Proceedings of 2011 Annual Computer Security Applications Conference (ACSAC'11)*, December 2011.

[118] Muhammad Naveed, Manoj Prabhakaran, and Carl A Gunter. Dynamic searchable encryption via blind storage. In *Proceedings of S&P'14*. IEEE, 2014.

[119] Tsuen-Wan "Johnny" Ngan, Roger Dingledine, and Dan S. Wallach. Building Incentives into Tor. In *Proceedings of Financial Cryptography*, January 2010.

[120] Huansheng Ning, Hong Liu, and Laurence T Yang. Aggregated-proof based hierarchical authentication scheme for the internet of things. *IEEE Transactions on Parallel and Distributed Systems*, 26(3):657–667, 2015.

[121] Lasse Øverlier and Paul Syverson. Locating hidden servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, May 2006.

[122] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[123] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal verification of operations on dynamic sets. In *Advances in Cryptology*. Springer, 2011.

[124] Darren Pauli. Hacked terminals capable of causing pacemaker deaths. https://www.itnews.com.au/news/hacked-terminals-capable-of-causing-pacemaker-mass-murder-319508.

[125] RAPID7. Baby monitor exposures and vulnerabilites. https://information.rapid7.com/iot-baby-monitor-research.html.

[126] Michael Reiter and Aviel Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), June 1998.

[127] Yonglin Ren, Richard Werner Nelem Pazzi, and Azzedine Boukerche. Monitoring patients via a secure and mobile healthcare system. *Wireless Communications, IEEE*, 2010.

[128] SAS. Iot security. https://www.sas.com/en_us/insights/articles/risk-fraud/iot-security.html#.

[129] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[130] Zach Shelby and Carsten Bormann. *6LoWPAN: The wireless embedded Internet*, volume 43. John Wiley & Sons, 2011.

[131] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). 2014.

[132] Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In *Proceedings of ESORICS 2006*, September 2006.

[133] Robin Snader and Nikita Borisov. A tune-up for Tor: Improving security and performance in the Tor network. In *Proceedings of the Network and Distributed Security Symposium - NDSS '08*, February 2008.

[134] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical technizheng ues for searches on encrypted data. In *Proceedings of S&P'00*. IEEE, 2000.

[135] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *Proceedings of NDSS'14*, 2014.

[136] Erik E Stone and Marjorie Skubic. Fall detection in homes of older adults using the microsoft kinect. *IEEE journal of biomedical and health informatics*, 19(1):290–301, 2015.

[137] Wenhai Sun, Xuefeng Liu, Wenjing Lou, Y Thomas Hou, and Hui Li. Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data. In *Proceedings of INFOCOM'15*. IEEE, 2015.

[138] Chiu C Tan, Haodong Wang, Sheng Zhong, and Qun Li. Ibe-lite: a lightweight identity-based cryptography for body sensor networks. *IEEE Trans. Inf Technol Biomed*, 13(6):926–932, 2009.

[139] Can Tang and Ian Goldberg. An improved algorithm for Tor circuit scheduling. In *Proc. of the 2010 ACM Conf. on Computer and Communications Security*, 2010.

[140] Yue Tong, Jinyuan Sun, Sherman SM Chow, and Pan Li. Cloud-assisted mobile-access of health data with privacy and auditability. *IEEE J. Biomed Health Inform*, 18(2):419–429, 2014.

[141] TorProject. Estimated Number of Clients in the Tor Network. `https://metrics.torproject.org/clients-data.html`.

[142] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *USENIX Security*, 2016.

[143] Jaideep Vaidya and Chris Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215. ACM, 2003.

[144] Jaideep Vaidya and Chris Clifton. Privacy-preserving decision trees over vertically partitioned data. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 139–152. Springer, 2005.

[145] Thijs Veugen. Comparing encrypted data. *Multimedia Signal Processing Group, Delft University of Technology, The Netherlands, and TNO Information and Communication Technology, Delft, The Netherlands, Tech. Rep*, 2011.

[146] John Villasenor. Five lessons on the 'security of things' from the jeep cherokee hack. `https://www.forbes.com/sites/johnvillasenor/2015/07/27/five-lessons-on-the-security-of-things-from-the-jeep-cherokee-hack/#692a18b4692a`.

[147] Cong Wang, Ning Cao, Jin Li, Kui Ren, and Wenjing Lou. Secure ranked keyword search over encrypted cloud data. In *Proceedings of ICDCS'10*, 2010.

[148] He Wang, Ted Tsung-Te Lai, and Romit Roy Choudhury. Mole: Motion leaks through smartwatch sensors. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 155–166. ACM, 2015.

[149] Tao Wang, Kevin Bauer, Clara Forero, and Ian Goldberg. Congestion-aware Path Selection for Tor. In *Proc. of Financial Cryptography and Data Security*, 2012.

[150] Tao Wang, Kevin Bauer, Clara Forero, and Ian Goldberg. Congestion-aware path selection for tor. *Financial Cryptography and Data Security*, pages 98–113, 2012.

[151] Xinyuan Wang, Shiping Chen, and Sushil Jajodia. Network flow watermarking attack on low-latency anonymous communication systems. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 116–130. IEEE, 2007.

[152] Tim Winter. Rpl: Ipv6 routing protocol for low-power and lossy networks. 2012.

[153] Rebecca Wright and Zhiqiang Yang. Privacy-preserving bayesian network structure computation on distributed heterogeneous data. In *Proceedings of the tenth ACM SIGKDD*

*international conference on Knowledge discovery and data mining*, pages 713–718. ACM, 2004.

[154] David J Wu, Tony Feng, Michael Naehrig, and Kristin Lauter. Privately evaluating decision trees and random forests. *Proceedings on Privacy Enhancing Technologies*, 4:1–21, 2016.

[155] David J Wu, Ankur Taly, Asim Shankar, and Dan Boneh. Privacy, discovery, and authentication for the internet of things. In *European Symposium on Research in Computer Security*, pages 301–319. Springer, 2016.

[156] Jacob Wurm, Khoa Hoang, Orlando Arias, Ahmad-Reza Sadeghi, and Yier Jin. Security analysis on consumer and industrial iot devices. In *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*, pages 519–524. IEEE, 2016.

[157] Lei Yang, Abdulmalik Humayed, and Fengjun Li. A multi-cloud based privacy-preserving data publishing scheme for the internet of things. In *Proceedings of ACSAC' 2016*. ACM, 2016.

[158] Lei Yang and Fengjun Li. Detecting false data injection in smart grid in-network aggregation. In *Smart Grid Communications, IEEE Conference on*. IEEE, 2013.

[159] Lei Yang and Fengjun Li. Enhancing traffic analysis resistance for tor hidden services with multipath routing. In *Communications and Network Security (CNS), 2015 IEEE Conference on*, pages 745–746. IEEE, 2015.

[160] Lei Yang and Fengjun Li. mtor: a multipath tor routing beyond bandwidth throttling. In *Communications and Network Security (CNS), 2015 IEEE Conference on*, pages 479–487. IEEE, 2015.

[161] Lei Yang and Fengjun Li. mTor: A Multipath Tor Routing Beyond Bandwidth Throttling. In *Communications and Network Security (CNS), 2015 IEEE Conference on*. IEEE, 2015.

[162] Lei Yang, Hao Xue, and Fengjun Li. Privacy-preserving data sharing in smart grid systems. In *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*, pages 878–883. IEEE, 2014.

[163] Lei Yang, Qingji Zheng, and Xinxin Fan. RSPP: A Reliable, Searchable and Privacy-Preserving e-Healthcare System for Cloud-Assisted Body Area Networks. In *INFOCOM, 2017 Proceedings IEEE*. IEEE, 2010.

[164] Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

[165] Qingji Zheng, Shouhuai Xu, and Giuseppe Ateniese. Vabks: verifiable attribute-based keyword search over outsourced encrypted data. In *INFOCOM'14*.

[166] Jun Zhou, Zhenfu Cao, Xiaolei Dong, Naixue Xiong, and Athanasios V Vasilakos. 4s: A secure and privacy-preserving key management scheme for cloud-assisted wireless body area network in m-healthcare social networks. *Information Sciences*, 314:255–276, 2015.

[167] Zhibin Zhou and Dijiang Huang. Efficient and secure data storage operations for mobile cloud computing. In *Network and Service Management*, 2012.