EPFL

Thèse n° 7206

Analyzing and Protecting Communication Metadata

Présentée le 17 septembre 2021

Faculté informatique et communications
Laboratoire pour la Sécurité des Données
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Ludovic BARMAN

Acceptée sur proposition du jury

Prof. M. J. Payer, président du jury
Prof. J.-P. Hubaux, Prof. B. A. Ford, directeurs de thèse
Prof. C. Diaz, rapporteuse
Dr N. Taft, rapporteuse
Prof. C. Troncoso, rapporteuse

École
polytechnique
fédérale
de Lausanne

2021

Contents

1 Introduction **7**

2 Metadata Leakage from Communications: Wearable Devices **11**
2.1 Introduction . . . 12
2.2 Background . . . 14
2.3 System and Adversarial Model . . . 15
2.4 Methodology and Datasets . . . 17
2.5 Device Identification . . . 20
2.6 Action and Application Identification . . . 22
2.7 Protections . . . 33
2.8 Discussion & Limitations . . . 38
2.9 Related Work . . . 41
2.10 Conclusion . . . 42

3 Reducing Metadata Leakage from Static Objects **43**
3.1 Introduction . . . 43
3.2 Motivation and Applications . . . 45
3.3 PURB Construction . . . 45
3.4 Padmé Construction . . . 46
3.5 Evaluation . . . 50
3.6 Related Work . . . 54
3.7 Limitations . . . 54
3.8 Conclusion . . . 55

4 Reducing Metadata Leakage from Communications **57**
4.1 Introduction . . . 57
4.2 Background . . . 58
4.3 Related Work . . . 60
4.4 PriFi: Anonymous Communications for Local-Area Networks . . . 62
4.5 Rubato: Large-Scale Asynchronous Anonymous Messaging . . . 81
4.6 Conclusion . . . 101

5 Conclusion **103**

Bibliography **107**

A Appendix: Wearable Devices (Chapter 2) **115**

B Appendix: PriFi (§4.4) **125**

C Appendix: Rubato (§4.5) **137**

Abstract

Most communication systems (*e.g.*, e-mails, instant messengers, VPNs) use encryption to prevent third parties from learning sensitive information. However, encrypted communications protect the contents but often leak metadata: the amount of data sent and the time it was sent, the way the data should be decrypted, the identity of the sender and the recipient. These metadata are a pervasive threat to privacy: They enable a variety of attacks that range from recovering plaintext contents from encrypted communications to inferring communicating parties.

Our goal in this thesis is two-fold: First, to raise awareness about this problem by demonstrating attacks that threaten user privacy; and second, to propose novel solutions that reduce the metadata leakage and maintain acceptable usability and costs.

To achieve the first goal, we present the first work that performs an in-depth analysis of the communications of wearable devices under the lens of traffic analysis. We demonstrate that the metadata of Bluetooth communications of wearable devices (smartwatches, fitness trackers, and blood-pressure monitors) leak sensitive information to a passive observer, despite the use of encryption. By design, these devices handle fine-grained and long-term, personal, medical and lifestyle-related data from their users. We show that typical defense strategies are ineffective: they only moderately hamper the adversary's task and have a high overhead. Our work highlights the need to rethink how sensitive data is exchanged in this setting. More generally, we confirm that metadata can pose a threat to user privacy, even in settings where traffic-analysis attacks are perhaps not an immediate threat.

For the second goal, we begin by presenting a theoretical contribution that concerns ciphertext formats: Padmé, a padding function designed to reduce the metadata leakage of files and messages through their length. Padmé efficiently hides the size of objects, even when they have very different sizes. Padmé is a part of PURBs, a ciphertext format that does not leak metadata, except for a small amount about the size.

Then, we design systems that enable communicating while reducing metadata leakage: Anonymous Communication Networks (ACNs). ACNs protect against some metadata leakage (*e.g.*, who is communicating and when). Traffic-analysis resistant ACNs have not seen widespread adoption yet, possibly due to their technical shortcomings. We focus on two particular aspects that remain unsolved by the related work: small-scale traffic-agnostic communication that achieves low latency, and large-scale asynchronous messaging that handles millions of users.

We present PriFi, an ACN that provides provable traffic-analysis resistance and low-latency in the context of a local-area network. The protocol hides the source of a message by ensuring that the communication patterns of all participants are equal, even in the presence of active attacks. PriFi has a high bandwidth cost but ensures low-latency and traffic-agnostic communication for a small set of users.

We then present Rubato, an ACN for anonymous messaging that handles millions of users. Other large-scale ACNs have an important limitation: A sender and a recipient need to be online at the same time to resist intersection attacks. As participants are uncoordinated, they need to be online and to send cover traffic at all times, which is infeasible for many users. We present Rubato, a circuit-based mixnet that enables its users to be asynchronous; they can participate in the network according to their own schedule. Unlike all previous ACNs, this enables users to participate using their mobile devices.

Résumé

La plupart des systèmes de communications (*e.g.,* courriels, messageries instantanées, VPNs) font appel au chiffrage pour empêcher qu'une personne non-autorisée n'accède à des données confidentielles. Malheureusement, les communications chiffrées protègent le contenu mais laissent échapper des informations au travers des métadonnées : les volumes de données et le moment de leur émission, la façon dont les données chiffrées doivent être manipulées, l'identité de l'émetteur et du destinataire. Ces métadonnées sont une menace insidieuse pour la protection des données privées: elles permettent diverses attaques dites «d'analyse de trafic», qui peuvent révéler qui sont les participants d'une communication privée ou même certaines données «en clair» malgré le chiffrage.

L'objectif de cette thèse est double: premièrement, de susciter une prise de conscience en démontrant une attaque qui menace la vie privée d'utilisateurs; deuxièmement, de proposer de nouvelles solutions qui réduisent les métadonnées révélées tout en maintenant un coût acceptable.

Pour atteindre le premier objectif, nous présentons la première analyse en profondeur des métadonnées des communications des *wearables* (appareils connectés portés par un utilisateur, *e.g.,* montres connectées, fitness trackers, moniteurs de pression cardiaque). Ces appareils traitent avec des informations médicales et personnelles liées au style de vie de l'utilisateur, sur le long terme et avec une granularité fine. Nous démontrons que les métadonnées des communications Bluetooth de ces appareils révèlent des informations personnelles et sensibles à un observateur passif, malgré l'utilisation du chiffrage. Par ailleurs, nous démontrons que les mécanismes de défenses typiques ne sont pas efficaces: ils ne réduisent que partiellement l'efficacité de l'attaque et ont un coût élevé. Notre contribution souligne le besoin de repenser globalement à la manière dont ces données sont échangées dans ce contexte. De manière plus générale, ceci confirme que les métadonnées peuvent constituer une menace pour la vie privée des utilisateurs, même dans un contexte où l'analyse des métadonnées n'est pas pratique courante.

Concernant le deuxième but, nous commençons par une contribution théorique. Nous présentons Padmé, une fonction de remplissage[1] conçue pour réduire les métadonnées qui sont révélées par la taille d'un contenu. Padmé masque efficacement les tailles de contenus, même si elles sont de grandeurs très différentes. Padmé fait partie de PURBs, un format de données chiffrées qui ne révèlent aucune métadonnée, sauf une quantité contrôlée de métadonnées via la taille.

Puis, nous nous concentrons sur la création de systèmes qui permettent de communiquer en limitant les métadonnées révélées: les réseaux de communications anonymes (RCA). Les RCAs dissimulent certaines métadonnées, par exemple l'identité des acteurs d'une communication,

[1] En anglais: «Padding function».

et le moment où ils transmettent des données. À ce jour, les RCAs les plus sécurisés n'ont pas connu un usage étendu, possiblement à cause de limitations techniques. Nous nous concentrons sur deux aspects non-résolus: les communications anonymes à haute fréquence et à petite échelle, et les communications anonymes à large échelle.

Pour garantir des communications anonymes et rapides au niveau des réseaux locaux, nous proposons PriFi, un RCA à haute fréquence et à l'épreuve des analyses de trafic. Ce protocole protège les émetteurs de messages en assurant que leurs communications suivent des motifs identiques, même en présence d'attaques actives. PriFi a un coût élevé en bande passante, mais permet des communications à faible latence pour un petit nombre d'utilisateurs.

Nous proposons ensuite Rubato, un système de messagerie instantanée anonyme qui supporte des millions d'utilisateurs. Les systèmes actuels similaires souffrent d'un important défaut: un émetteur et un destinataire doivent être connectés au même moment pour résister à certains types d'attaques. Si les participants ne peuvent pas se coordonner, cela implique qu'ils doivent être connectés et envoyer des messages factices en permanence. Cette contrainte est souvent irréalisable pour la plupart des utilisateurs. D'autre part, les téléphones mobiles sont de facto exclus de ce type de protocole d'échange. Rubato est un réseau de mixage qui permet à ses utilisateurs d'être asynchrones et non-coordonnés, et donc de bénéficier de communications anonymes selon leur propre horaire. Contrairement aux systèmes actuels, Rubato permet des communications anonymes depuis des téléphones portables.

Acknowledgements

This thesis would not have been possible without the help, guidance, and contributions of many friends and collaborators.

First, I am very grateful to my advisors Prof. Jean-Pierre Hubaux and Prof. Bryan Ford. Both contributed immensely to this thesis taking shape. I thank them especially for their patience, guidance, and their uninterrupted support through the ups and downs of research, and for the total freedom they gave me in choosing my topics. Looking back, I enjoyed greatly these years at LDS and DEDIS. Thank you for this opportunity.

I wish to thank my jury committee, Prof. Claudia Diaz, Prof. Mathias Payer, Dr. Nina Taft and Prof. Carmela Troncoso for finding the time in their busy schedules to review this dissertation.

I am very thankful to all my colleagues and friends at LDS and DEDIS for making these years the pleasant, thrilling, and fun experience they have been. I would like to thank my colleagues at LDS: Alexandra, Christian, David, Francesco, Jean-Philippe, Jean-Louis, Joao, Juan, Romain and Sinem. I also wish to thank the first-generation of DEDIS: Cey, Gaurav, Gaylor, Henry, Jeff, Kelong, Lefteris, Linus, Maggy, Nicolas, Noémien, Philipp, and Simone, along with the relief team: David, Haoqian, Louis-Henri, Pasindu, Pierluca and Sandra. Thank you all for the interesting discussions, the fancy restaurants, and the weekend trips.

I wish to thank Angela and Patricia for their invaluable help with the administrative tasks through the years. I also thank Holly for her relentless efforts with the editing of my texts.

The technical contributions of co-authors and collaborators are acknowledged in the relevant chapters. More importantly, I wish to thank them all for our very pleasant collaborations. I will remember these times of joint efforts towards a shared goal as the most enjoyable parts of my PhD.

Finally, a special word goes to Anh, Apostolos, Italo, Kirill and Sylvain for being excellent partners in crime.

1. Introduction

The modern era of information technology is characterized by a rapid integration of digital technologies into virtually all aspects of society. In particular, individuals increasingly use a plethora of connected devices: instant messengers to communicate, IoT devices to secure their homes, and wearable devices to measure vitals, fitness activities, and sleep patterns. Never before in history has such a large quantity of personal data been recorded, sent, shared and analyzed [124].

However, all the benefits brought by the processing of personal data come with an increase of the threats to individual privacy and security. Personal information is extremely valuable to many entities: Advertisers build profiles from user data [85, 105]; companies monitor and track users [39, 64, 117, 156, 242]; hackers can blackmail users and steal identities [198, 214]; governments monitor and track citizens by collecting and sifting over large amounts of data [37, 130, 241].

Thankfully, this looming threat is partially contrasted by the public's slowly-increasing awareness of risks and their rights to digital privacy. Interest in encrypted communication rose after the 2013 Snowden leaks about mass surveillance [123]. Perhaps one of the consequences is that the use of encryption on the Web has seen a rapid increase over the past decade [106]. Similarly, in the Western world, we observed an increased adoption of end-to-end encryption in instant messengers [15, 197, 230].

This shift towards encrypted communications is a necessary step towards privacy: it protects data in transit from eavesdroppers. However, despite the strong properties provided by encryption, such an eavesdropper might still be able to infer sensitive information through communication *metadata*: contextual information such as the identity of the sender and the recipient, the amount of data sent and the time at which it was sent, the way the encrypted data should be handled. As they are a part of the context hence often not encrypted in the data, such metadata are readily available to third parties.

Communication metadata can be surprisingly informative. Two decades of research demonstrate how they can be (mis)used: The metadata of encrypted VoIP calls enable the inference of the spoken language and some sentences, despite the use of encryption [46, 236]; the metadata of anonymized phone and VoIP calls enables the re-identification and geolocation of users [147, 155]. Through the analysis of communication metadata, web activities on a VPN are identifiable despite encryption [79, 201], and this is also the case for video streams over TLS [174, 175, 189] and Web pages over Tor [164, 226, 227]. By using communication metadata, user activities can be inferred from encrypted communications in a smart home [2, 204] and on smartphones [60, 186, 207].

But the analysis of communication metadata is not only a research exercise: In 2013, Edward Snowden revealed how the National Security Agency (NSA) was conducting, as part of their

intelligence operations, a large-scale collection of metadata. The fact that metadata were collected — and not data — is telling. When commenting on the issue, the former NSA General Counsel Steward Baker said,

> "Metadata absolutely tells you everything about somebody's life. If you have enough metadata, you don't really need content." Steward Baker [56, ¶2]

These examples indicate that in the wrong hands, communication metadata can be a threat to privacy: They can be sensitive in themselves, and they can reveal information from encrypted communications. We deem that the threat is both pervasive and concrete: Despite these leaks and the many research works highlighting the problem, in practice, few systems enable users to hide some of their communication metadata [78]. Virtually all widespread communication systems (*e.g.,* Signal, WhatsApp, e-mails, VPNs) leak some metadata to a network observer. Some of these metadata are available to many private and public actors, some of whom have incentives to collect data: *e.g.,* companies that provide end-to-end instant messaging services, employers, ISPs, and governments.

Hence, our goal in this thesis is two-fold: first, to raise awareness about this problem by demonstrating attacks that threaten user privacy; second, to propose novel solutions that reduce the metadata leakage of encrypted communications.

Scope. We note that many researchers are working on the open problem of protecting communication metadata [19, 102, 201]. Our goal in this thesis is not to "solve" the problem as a whole; we believe that a holistic solution might not even exist. Rather, we propose novel solutions to specific problems in order to advance the state of the art, in the hope to see, one day, better privacy and security practices in digital communications.

Contributions

Our first goal is to raise awareness about the problem of communication metadata. Although the problem of information leakage through metadata is now well-acknowledged in the security research community, most real-world systems still leak sensitive communication metadata. One hypothesis explaining the current state of affairs outside the research world is that the problem is sometimes considered to be a remote concern, or to be too costly to solve compared to the concreteness of the threat. We seek to demonstrate the problem in a sensitive, previously unexplored setting. We use for an example the ecosystem of wearable devices: smartwatches, step counters, blood pressure monitors, sleep trackers, etc. We demonstrate that the Bluetooth communications of wearable devices can leak sensitive information to a passive observer, despite the use of encryption. By design, these devices handle fine-grained and long-term personal, medical and lifestyle-related data. We explore common defense strategies that moderately hamper the adversary's task, yet at a high cost. This highlights the need to rethink how sensitive data is exchanged in this setting.

Our second goal is to propose new solutions to the problem of metadata leakage in communications. As addressing the overarching problem of communication metadata is notoriously difficult, we focus on three particular aspects: (1) a ciphertext format that reduces metadata leakage, (2) a system for small-scale traffic-agnostic communications that achieves low latency, and (3) a system for large-scale asynchronous messaging that handles millions of users.

First, we consider a single static file or message, such as a PGP ciphertext. To ease decryption, such an encrypted file typically reveals many metadata: the full list of public keys it has

been encrypted for, the cryptographic parameters used, and its length. We[1] propose a new ciphertext format that does not leak any metadata at all, except for its length, for which the leakage is minimized. In this thesis, we introduce Padmé, a padding function designed to reduce the metadata leakage of files and of messages through their length. Padmé efficiently hides the size of objects, even when the sizes vary significantly.

Second, we focus on the design of systems that communicate without leaking metadata: anonymous communication networks (ACN). ACNs protect against some metadata leakage, such as the identity of the person communicating. Traffic-analysis resistant ACNs have not seen widespread adoption, possibly due to their technical shortcomings.

We first focus on the setting of organization networks: LANs and WLANs in companies, universities, and non-governmental organizations. It is easy for a local eavesdropper to capture traffic and launch targeted attacks against high-value individuals. Current ACNs are poorly suited to hiding communication metadata in the context of an organizational network; most of them rely on mixing or shuffling over a chain of servers distributed around the world, which adds latency. We present PriFi, an ACN that provides provable traffic-analysis resistance and low-latency in the context of a local-area network. The protocol hides the communicating entities by ensuring that the communication patterns of all participants are equal, even in the presence of active attacks. PriFi has a high bandwidth cost, but ensures low-latency, traffic-agnostic communication for a small set of users.

Finally, we focus on large-scale anonymous messaging. A fundamental limitation of current large-scale ACNs is *synchronicity*: users need to follow a fixed global schedule to resist intersection attacks. In practice, this means that users need to be online and send cover traffic at all times, which is unrealistic for many users. In particular, this constraint is hardly compatible with mobile devices. We present Rubato, an ACN for anonymous messaging that handles millions of users and enables its users to be asynchronous, that is to say, to participate in the network according to their own schedule. This enables users to choose their quality of service, depending on their own constraints such as battery life. At the same time, all users benefit from a single global anonymity set.

Thesis Outline

In Chapter 2, we demonstrate how communication metadata can be used to infer sensitive information from wearable devices. In Chapter 3, we present PURBs and Padmé, respectively, a ciphertext format and a padding function. In Chapter 4, we present PriFi and Rubato, two anonymous communication networks. We conclude and present the limitations of our work in Chapter 5.

Publications

Chapter 2 is based on the results of [23]. Chapter 3 is based on the results of [161]. Chapter 4 is based on the results of [20, 25, 26].

[1] This contribution is a joint work with Kirill Nikitin; the contribution is separated between the two theses (Chapter 3).

2. Metadata Leakage from Communications: Wearable Devices

In this first chapter, we study the effect of metadata leakage in one particular ecosystem in which data privacy is essential: wearable devices. Devices such as smartwatches, fitness trackers, and blood-pressure monitors have access to long-term and fine-grained sensitive information about the wearer: health-related data, fitness-related metrics, daily usage patterns, etc. This personal data is typically synchronized with a companion app running on a smartphone over a Bluetooth (Classic or Low Energy) connection.

In this work, we investigate the information that can be inferred from the metadata (such as the packet timings and sizes) of encrypted Bluetooth communications between a wearable device and its connected smartphone. We consider a nearby Bluetooth eavesdropper (*e.g.,* a nosy neighbor, a smart billboard in a shop). We demonstrate that such an adversary can use traffic patterns to recognize the devices, the applications and the actions performed by the users in its vicinity.

In particular, we show that a passive eavesdropper can use communication metadata to accurately recognize:

1. communicating devices, even without having access to the MAC address or friendly name;
2. human actions (*e.g.,* monitoring heart rate, exercising) performed on wearable devices ranging from fitness trackers to smartwatches;
3. the mere opening of specific applications on a Wear OS smartwatch (*e.g.,* the opening of a medical app, which can immediately reveal a condition of the wearer);
4. fine-grained actions (*e.g.,* recording an insulin injection) within a specific application that helps diabetic users to monitor their condition;
5. the profile and habits of the wearer, by continuously monitoring their traffic over an extended period.

We run traffic-analysis attacks by collecting a dataset of Bluetooth communications concerning a diverse set of wearable devices, by designing features based on packet sizes and timings, and by using machine learning to classify the encrypted traffic to actions performed by the wearer. Then, we explore standard defense strategies against traffic-analysis attacks such as padding, delaying packets, or injecting dummy traffic. We show that these defenses do not provide sufficient protection against our attacks and introduce significant costs. Overall, our research highlights the need to rethink how applications exchange sensitive information over Bluetooth, to minimize unnecessary data exchanges, and to research and design new defenses against traffic analysis tailored to the wearable setting.

Acknowledgments. This work has been made possible by the help of Alexandre Dumur, Friederike Groschupp, and Stéphanie Lebrun. It is a joint work with Dr. Apostolos Pyrgelis.

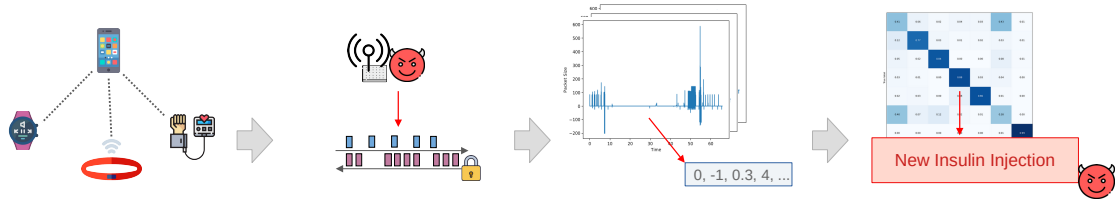


Figure 2.1 – Traffic-analysis attack on the encrypted traffic of Bluetooth wearable devices. After an offline training phase (not depicted here), the passive eavesdropper can recognize the action to record an insulin injection despite the use of encryption.

2.1 Introduction

With the rising interest in personalized health care and "quantified self", wearable Bluetooth devices are becoming pervasive in our societies. To improve aspects of their daily lives, users increasingly use smartwatches, medical and fitness-related devices such as activity trackers, step counters, blood-pressure monitors and sleep trackers[1]. These devices process, store, and transmit personal and sensitive data linked to the wearer's identity and health status: fine-grained and long-term activity levels, heart rates, arrhythmia alerts, medication and sleep schedules, etc. Such personal information should be protected from third parties, as it can be used to build profiles, to identify and track users (*e.g.*, for advertising purposes), or can be sold to insurance companies for the quantification of users' medical risks. Medical wearable devices that are FDA-approved[2] are already subject to such a requirement: they "should have appropriate protections in place that prevent sensitive information from being read by unauthorized parties either in storage or in transmission" [88].

For enhanced functionalities, most wearable devices communicate with a smartphone via Bluetooth. These devices can use encryption at the Bluetooth link layer or application layer. In this work, we show that these encrypted communications leak information about their contents via their communication patterns (*e.g.*, distribution of packet sizes and inter-arrival timings). We consider a local passive eavesdropper who attempts to infer sensitive information from the communications. This adversary could be a nosy neighbor, an advertiser in a shopping mall attempting to infer the shoppers' habits, an employer [64], or a more nefarious adversary collecting data for sale to insurance companies. These threats against privacy are concrete: For over a decade, advertisers have been using Bluetooth, Wi-Fi, and cellular networks, to track consumers in stores [117, 156, 242], and to link their profiles to online advertisement databases [150]; these advertisers could use Bluetooth traffic patterns to learn new information or to better profile users.

We infer sensitive information from the encrypted communications of wearable devices by using *traffic-analysis attacks* [66], a technique that exploits the communication patterns (*e.g.*, packet sizes and timings) of encrypted traffic. These attacks have been successfully demonstrated in diverse settings: to recognize web pages on Tor traffic [82, 164, 226, 227], to fingerprint devices [165] or to infer the activities performed in a user's smart home [2, 204] and to recognize user activities and applications used on a smartphone (*e.g.*, sending an e-mail or browsing a web page) [60, 186, 207, 208, 244]. The focus of recent related works has been on inferring user activities in the IoT and smart home setting [2, 6, 16, 17, 215], eavesdropping on

[1] In 2019, 46 million wearable devices were sold. This number is expected to rise to 158 millions by 2022 [205].
[2] U.S. Food and Drug Administration. This institution notably edicts rules for medical devices sold in the U.S.

WLAN or Internet traffic (or both). Very few related works consider the communications of Bluetooth wearable devices at the Personal Area Network (PAN) scale, despite the sensitive nature of the information exchanged. Das *et al.* [69] demonstrate how the bitrate of some fitness trackers is correlated with the user's gait; however, their analysis is restricted to 6 BLE devices, and they do not attempt to recognize devices, applications or users actions. To the best of our knowledge, ours is the first work that performs in-depth device, application and user-action identification on the Bluetooth communications of wearable devices.

To perform the traffic analysis of wearable devices, we build a Bluetooth (Classic and Low Energy) traffic-collection framework. We set up a testbed consisting of a diverse set of wearable devices (smartwatches, fitness trackers, blood-pressure monitors, etc.) connected to a smartphone, and we use a Bluetooth sniffer to capture the traffic. To generate realistic traffic traces, we manually use the wearable devices in the intended way: *e.g.,* by performing a short running exercise in the case of a fitness monitor, repeated multiple times to obtain a sufficient number of samples. We obtain a dataset of labeled (encrypted) Bluetooth Classic and Low Energy traces; we then design features based on packet sizes and timings, and we apply machine-learning classification techniques for identifying devices, applications, and user actions despite the encryption.

Our experimental results show that an eavesdropper can exploit encrypted communication patterns to passively identify devices, even when these devices are already paired and do not broadcast Bluetooth addresses or names. Our methods identify specific models/versions of the device, and hence enable the tracking of devices and users, despite the Bluetooth address-randomization protection-mechanism employed by the Bluetooth Low-Energy protocol. Moreover, we demonstrate that the adversary, for instance a nosy neighbor, can use our traffic-analysis attacks to accurately recognize user actions (*e.g.,* recording the heart rate, beginning a workout or receiving an SMS) performed on different wearable devices such as smartwatches, step counters, and fitness trackers. We then focus on smartwatches and show that an eavesdropper, for instance an unscrupulous advertiser, can recognize the mere opening of specific applications, which can be immediately sensitive (*e.g.,* in the case of a medical app) or used to build a profile (*e.g.,* based on religious or political apps). Furthermore, we show that our methodology generalizes well: The model trained on a smartphone/wearable device pair performs accurately on a different device pair, indicating its cost-effectiveness. We also highlight how a targeted adversary can recognize sensitive user activities within a specific application; we accurately recognize the action of recording an insulin injection in a diabetes-helper application (Figure 2.1). Finally, we show that an attacker can infer a wearer's habits and build their profile by eavesdropping her Bluetooth traffic over a long time-period.

Finally, we focus on countermeasures against the presented traffic-analysis attacks. We evaluate how standard approaches against traffic analysis (*i.e.,* padding or delaying messages, injecting dummy traffic) perform against our attacks. Our results show that these defenses provide insufficient protection: though they yield a drop in the adversary's accuracy, none of them reduces it to that of random guessing. Furthermore, their costs are high (from tens to hundreds of kilobytes of additional exchanged data) for wearable device communications where energy consumption is crucial. We also observe that the effectiveness of these defenses varies greatly with the adversarial task: although the "right" defense drops the attacker's accuracy, non-adapted defenses have almost no effect yet still incur high costs. This suggests that a "one-size fits all" defense for preventing device fingerprinting, as well as application and action fingerprinting, is unlikely to exist at a reasonable cost. Our defense evaluation highlights the need to rethink how wearable devices exchange sensitive data over Bluetooth

communications and prompts for the design of novel defenses. For example, for applications that can support it, *data minimization* is a valid strategy: data that is not exchanged cannot be fingerprinted, and we observe in our experiments that low-volume exchanges are naturally protected from traffic analysis. Moreover, bulk transfers (*e.g.,* synchronizing a step counter every day at midnight) also hamper the task of the adversary.

Overall, the purpose of our work is to raise awareness, notably among device manufacturers and application developers, of the limited confidentiality on the Bluetooth link with today's applications and devices. For instance, manufacturers might be willing to implement mitigations directly to the wearable devices' firmware, hence it is urgent that the research community provides them with acceptable solutions that will protect the next generation of wearable devices. Application developers might want to carefully consider the information their application can leak through its communication patterns and to implement an application-level defense (for instance, pad all communications to a fixed size). We hope that our research results will be a starting point for further research on the communication metadata of Bluetooth wearable devices. To facilitate future research on the topic, we open-source our dataset for research purposes: it consists of Bluetooth traces of wearable devices used to perform multiple actions and that have been, for most devices, manually recorded over months. Although our experiments focus only on Bluetooth communication metadata, our dataset contains all captured data (*e.g.,* link-layer information, pairings) that might be of independent interest.

In summary, our contributions in this chapter are as follows:

- We show traffic-analysis attacks that, based on the encrypted traffic of wearable devices, recognize:
  1. communicating wearable devices, up to the model number of the same device;
  2. user activities and the opening of specific applications;
  3. fine-grained sensitive actions (*i.e.,* recording an insulin injection) within an application;
  4. actions recorded over a long time-period, which can be used to build a profile.
- We experimentally evaluate standard defenses against traffic analysis and suggest possible strategies;
- We make available a large dataset of Bluetooth traffic captures for future research.

2.2 Background

There exist two flavors of Bluetooth: Bluetooth Classic, referred to as "BR/EDR" for Basic Rate/Enhanced Data Rate, and Bluetooth Low Energy (BLE). The former is used for data-intensive or latency-sensitive scenarios (*e.g.,* audio streaming), whereas the latter is used for low-power or low-throughput scenarios. Most wearable devices we collected for our testbed use Bluetooth Low Energy, except for smartwatches that typically use Bluetooth Classic. Both Bluetooth specifications [32] are produced by the Bluetooth Special Interest Group (SIG), and their latest version is 5.2.

Data Exchange. In both Bluetooth Classic and BLE, data exchange occurs after a pairing process between one or more slaves (the wearable device) and a master (*e.g.,* the smartphone). We remark that a smartwatch can be a master for another wearable device. Initially, to discover each other, devices broadcast and listen on *advertising channels* in Bluetooth Low-Energy, or on channels determined by a predefined hopping sequence in Bluetooth Classic. Then, the pairing process assigns the slave to the master's *Piconet*, and both devices communicate using

the non-advertising channels. Unlike Bluetooth Classic, BLE also supports data exchange without establishing a link-layer connection: devices simply broadcast short information to its surroundings on the advertising channel.

To communicate in a Piconet, all devices use the same frequency hopping pattern (derived from the master's MAC address and clock). The channel is divided into time-slots; odd time-slots are for the slaves, and even time-slots for the master. When the connection is Asynchronous Connectionless (ACL), devices communicate opportunistically during unreserved time-slots. In the case of Synchronous Connection-Oriented (SCO) connections, devices communicate at predetermined time-slots without acknowledgment.

Security. The security properties are similar between Bluetooth Classic and BLE. Devices first establish and authenticate a long-term key through a *pairing* process. In this process, both devices also derive a short- or long-term key. "Legacy" pairings are generally insecure by today's standards [235]: they rely on a short PIN and can be easily brute-forced. Secure Simple Pairing (SSP) is a more recent pairing protocol based on elliptic curve cryptography [169]. It is available since Bluetooth 2.1. There exists several active attacks on SSP, based on some form of Man-in-the-Middle [112, 113, 114] or low-entropy key generation [13, 14]. If the pairing is done correctly and the key uses sufficient entropy, the subsequent communication should be confidential and integrity-protected (AES-CCM with a 128-bit key).

Bluetooth Eavesdropping. Eavesdropping on Bluetooth traffic is complex due to the frequency hopping. Inexpensive sniffers (such as the Ubertooth [107]) attempt to follow the frequency hopping of a connection and often require observing the pairing [185]. However, by listening concurrently on all channels, high-end commercial scanners accurately capture all traffic without the need to observe the pairing. Recent advances in research use coordinated Ubertooth devices to achieve a greater capture accuracy [4, 5], or use software-defined radios (SDRs) to cover the Bluetooth spectrum at a cost lower than commercial scanners [57, 58]. We discuss the practicality of Bluetooth eavesdropping in §2.8.1.

2.3 System and Adversarial Model

We consider a user U that possesses a smartphone S and a collection of wearable devices W. This collection W contains heterogeneous devices: some with a general-purpose OS (*e.g.,* an Android smartwatch) or a simpler firmware (*e.g.,* a step counter). Devices in W can communicate with S over Bluetooth Classic or Low Energy.

Adversary. We consider an adversary A who is a passive eavesdropper. A uses a Bluetooth sniffer to capture all Bluetooth traffic in the vicinity of U. In particular, A might capture traffic from other users and devices that are also nearby. A does not compromise any devices and does not possess the keys to decrypt Bluetooth traffic.

Goals and Traffic Captured. Informally, A attempts to infer information from all communications between the wearable devices in W and S (Figure 2.2). However, we need to precisely define the aforementioned adversary. In practice, an eavesdropper who sees *all* traffic between a wearable device and its connected smartphone is not realistic: a sporadic or local adversary is unlikely to consistently capture one-time events such as pairings. Therefore, we define as A_{all} an adversary who sees all communications between a wearable-smartphone device pair, and we further define two weaker adversaries in the sense of A, but who only observe, respectively:

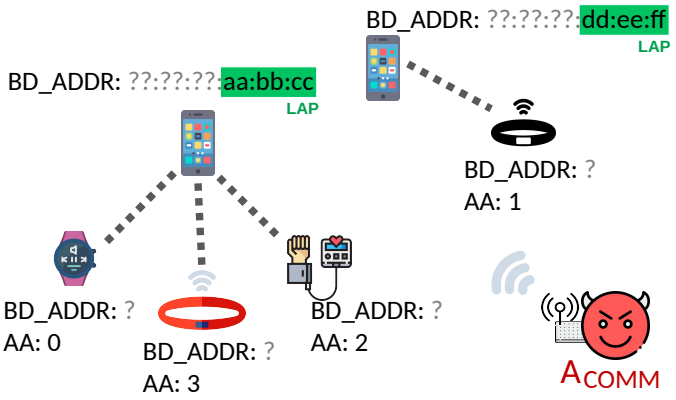


Figure 2.2 – Information visible to A_{comm}: (1) the active connections between each pair of wearable-smartphone, identified by a random Access Address (AA); A sees all packets on these connections; (2) for each master device, the Lower Address Part (LAP) of the BD_ADDR (sometimes called the MAC address). A_{comm} cannot tell what types of devices are communicating.

- A_{paging}: all paging events (devices waking up from sleep and performing minimal discovery) and subsequent communications, but not the pairings;
- A_{comm}: all active communications, but neither pairing nor paging events.

In terms of adversarial power, we have that $A_{\text{all}} \geq A_{\text{paging}} \geq A_{\text{comm}}$. We focus on the weakest adversary A_{comm} who only observes ongoing communications between a wearable device and its connected smartphone. This matches what a passive adversary can do consistently.

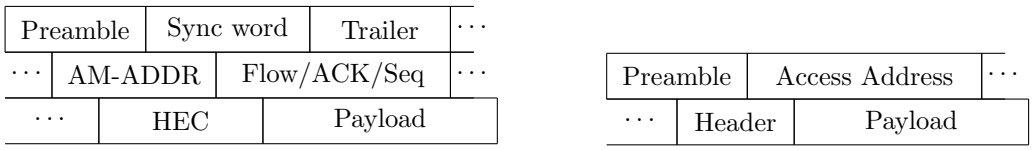
Information Visible to A_{comm}. Figure 2.3a shows the information that is visible to A_{comm} during Bluetooth Classic communications. The sensitive/identifying fields are the Sync word and the Active-Member Address (AM-ADDR). The Sync word reveals the 24 lower bits of the master's MAC address (LAP). The Upper Address Part (UAP), that is not transmitted in this packet, is assigned to manufacturers and identifies a wearable device. Finally, the Active-Member Address is a 3-bit integer (and not a MAC address) identifying a device in a given Piconet; this is similar to a connection identifier. Figure 2.3b shows the information visible to A_{comm} for Bluetooth LE. Similar to AM-ADDR, the Access Address is a connection identifier (and not a MAC address). On a higher-level, in both Bluetooth flavors, the information given to A_{comm} per packet is

$$(\text{Packet type}, \text{connection ephemeral ID}, \text{time}, \text{payload})$$

where the Packet type indicates whether Bluetooth Classic or LE is used, and where the connection ephemeral ID is randomized, except for the master device in Bluetooth Classic where it is fixed. The payload can contain upper-layer packet information, for instance the packet type, as well as the encrypted application payload.

Capabilities of A_{comm}. In conclusion, with the information visible on the Bluetooth channel, A_{comm} can

- enumerate the devices $w \in W$ and assign a pseudonym to each of them;
- group them by connection between a pair of devices; and
- collect a series $\{(\text{time}, \text{packet type}, \text{packet size})\}_w$ for each device $w \in W$ for the duration of the capture.



(a) Structure of a BB_PDU packet (BaseBand Packet Data Unit) used in Bluetooth Classic. HEC (Header Error Correction) is an error-correction code.

(b) Structure of a Link Layer packet used in Bluetooth LE.

Figure 2.3 – Packet Structures. The features used in our attacks are derived from the time of reception and the size of the payload.

We assume that the adversary is able to isolate a communication of interest, *e.g.,* using the Active-Member Address (AM-ADDR) of Bluetooth Classic or Access Address of Bluetooth Low Energy, possibly combined with other techniques, *e.g.,* distance estimation using the Received Signal Strength Indicator (RSSI). The pseudonymous recipient of a packet is often known, as most packets are acknowledged.

Attacker Goals. Given the Bluetooth information available, the goal of A_{comm} is threefold:

G1) **Device Identification:** for a device w, recognize the device brand/manufacturer;
G2) **Action Identification:** for a device w, recognize user actions (*i.e.,* interactions with the device);
G3) **Application Identification:** for a device w, recognize the running application.

We use Goal 1 as a stepping-stone for the following goals. However, meeting the first goal already has privacy implications: such an adversary can recognize and track a user through the list of her wearable devices, despite MAC address randomization, and can build a profile for that particular user.

2.4 Methodology and Datasets

We describe how the adversary defined in §2.3 can perform traffic-analysis attacks to achieve its goals. First, we build a data collection framework to record Bluetooth traffic and to automate the data transmission for some Bluetooth wearable devices. For other devices, we manually trigger Bluetooth traffic by performing the appropriate human action (*e.g.,* pressing a sequence of buttons, performing a physical activity). Then, we process the captured traffic and use it with machine-learning classification techniques to infer information from encrypted Bluetooth traffic.

Testbed. We set up a testbed with multiple wearable devices to account for a wide range of manufacturers, device capabilities, and functionalities. We initially selected 18 Bluetooth Classic and LE devices: 5 popular smartwatches, 2 headphones, and 11 other wearable devices (step counters, fitness trackers, and blood-pressure monitors) from the most popular vendors.

All Bluetooth Classic devices analyzed use link-layer encryption. To our surprise, from the Bluetooth Low Energy devices, only the Fitbit Charge 2 uses link-layer encryption. We perform entropy tests to validate that the remaining Low Energy devices use encryption at the application layer. We observe a high entropy (6 – 8bit of information per byte) for all devices, except for 3 of them (the `Beurer AS80`, `SW170`, and `PanoBike+`, ≈ 4bit/byte). Hence, we discard these three devices from our testbed. We remark that though our attacks do not use plaintext

contents and would work on these three devices, an attacker can achieve the same results by simply reading the payload. We also discard two blood-pressure monitors (the `Qardio` and `H2-BP`) that we could not reliably use. In total, the testbed consists of 13 devices: 7 Bluetooth Classic and 7 Low Energy devices (Tables 2.1 and 2.2).

Although our testbed consists of a modest number of devices, most devices use proprietary firmware and software (*i.e.,* they are closed-source), which does not allow us to automate their Bluetooth traffic-data collection. Except for Wear OS smartwatches that we can automate, the data collection is a sequential and manual process.

We use a Nexus 5 as the smartphone for all Android/Tizen smartwatches and wearable devices without an OS, and an iPhone 8 for the Apple Watch/AirPods. We updated all devices with the latest firmware and OS updates.

Actions Recorded. For each device, we manually compile a set of possible user-actions: low-end devices sometimes only `Sync` with the smartphone; mid-range devices support activities such as `Running Workout`, `Measure Heartbeat`, etc., whereas high-end devices such as smartwatches offer a large range of user actions through the use of additional apps (*e.g.,* `AppX Open`, `AppX DoActionY`) that users can choose to personalize their device.

Capture Methodology. We record all Bluetooth traffic by using a wide-band scanner (an Ellisys Vanguard [83]). For each device w, we pair w with the phone S and let some time pass to allow for an initial data synchronization. Then, we trigger the desired action and record the corresponding Bluetooth traffic for $T = 30$ seconds (we observe this to be a conservative value); this constitutes one sample. To collect sufficient number of samples for an action, we repeat the capture process $N = 25$ times (we observe this to be a conservative value, Appendix A.4, Figure A.1c). We record real activities: for instance, for a `Running Workout`, we perform a short running activity in the vicinity of the sniffer; for the wearable that records the heart rate, we fit the wearable to ourselves during the experiment. This is in contrast with some previous works (in the context of Wi-Fi traffic) that use UI fuzzing to quickly provide many traffic samples generated by smartphones [60, 207, 208]. This ensures that most values communicated by the wearable reflect what a real user would generate (*e.g.,* the number of steps, speed, and distance). However, due to our non-portable experimental setup, the recorded GPS coordinates will show an almost-fixed position.

Environment. We conduct the experiments in an office where both Wi-Fi devices and other Bluetooth devices communicate. This corresponds to a noisy environment; an anechoic chamber or a Faraday cage would advantage the adversary by allowing to record trace with less noise. We turned off unused devices to avoid unnecessary interference.

Dataset. We collect a dataset that consists of 10,700 Bluetooth captures with a duration of ≈30 sec, recorded over 13 devices, 80 applications and 32 user-actions (see Table 2.3). This amounts to ≈96 hours of recording and represents 21 GB of data. 10,371 of these traces are over Bluetooth Classic, and 329 Bluetooth Low Energy. 2,215 of these captures are the result of the corresponding human action (*e.g.,* performing a short workout), and 8,485 are automated actions (*e.g.,* the opening of an application) on Wear OS devices.

Additionally, we record a second dataset that contains longer captures with a duration of ≈ 20min. These captures contain automated actions that, when combined, represent a plausible usage of a smartwatch over a day. We use them to model a persistent adversary (§2.6.2.4). This amounts to 38 hours of recording and 9.5 GB of data.

Overall, the datasets contain raw binary files with all captured traffic. As the file format is proprietary, we extract the corresponding CSV files.

Sample Processing. We process the captured raw Bluetooth traces as follows. We manually select the connection of interest. We follow the approach of the related work and extract only the total length of the payload to avoid relying on the presence of plaintext markers in it [207, 208]. From each recorded Bluetooth trace, we extract the Bluetooth packets at the Logical Link Control and Adaptation Protocol (L2CAP) layer and output a series of (arrival time, size)-tuples. We label the trace with the `Device` used, the `Application` used (in the case of a smartwatch), the `Action` performed, and whether it used Bluetooth Classic or LE. We split the set of recorded samples S into two subsets S_{Classic} and S_{LE} depending on the Bluetooth variant used; An adversary is able to do the same, as the frames are different.

Feature Extraction. We design features to capture patterns based on incoming/outgoing size distributions and inter-packet timings. We map each sample in S_{Classic} and S_{LE} to a 32-scalar feature vector: First, we capture global statistics about packet sizes. We filter each sample into 3 packet sequences: (1) from Master to Slave, (2) from Slave to Master, and (3) sequences consisting of all non-null, non-ACK packets. From each filtered sequence, we extract the following 5 statistics: the min/mean/max/count/standard deviation of the packet sizes in bytes. Then, we observe that different devices/applications send packets with a distinctive size. To this end, following the techniques used by Liberatore and Levine [148] and Herrmann *et al.* [115] in the context of website fingerprinting, we extract features corresponding to the number of packets that are in a certain range. We select 10 buckets that represent the size (in bytes) ranges $[0,9], [10,19], \cdots, [80,89]$ and a last "catch-all" bucket for packets with $[90,\infty]$ bytes. Finally, we examine the timings of the packets. In more detail, we compute, in seconds, the series of inter-packet durations and extract the same 5 statistics (min/mean/max/count/standard deviation) from it. Furthermore, we calculate the average send/receive inter-packet times as done by Saltaformaggio *et al.* [186] in the case of TLS traffic: $\text{AvgIPT}(P) = \frac{\sum_{i=0}^{|P|-1} \text{ts}_{i+1} - \text{ts}_i}{|P|-1}$, where P is the set of sent/received packets, and ts_i is the timestamp of packet i. Intuitively, this captures the communication bursts of each device.

Classification. For our traffic-analysis attacks (described in §2.5 and §2.6), we use a Random Forest classifier, a popular algorithm for multi-class classification that is also widely used in the related work [207, 208]. We opt for a Random Forest classifier over the recently proposed deep-learning approaches [199, 200] due to its interpretability and the moderate size of our dataset. We split the samples of each device/application/action (depending on the adversarial goal) into 80% training and 20% testing sets and perform 10-fold random-stratified cross-validation. We also retain the most important features according to the classifier's importance using Recursive Feature Elimination (RFE).

Tables 2.1 and 2.2 — Bluetooth wearable devices used in our experiments. BT indicates the Bluetooth version. The `AppleWatch` uses both flavors of Bluetooth.

Table 2.1 – Bluetooth Classic devices.

Vendor	Model	OS	BT	Chipset
Samsung	Galaxy Watch	Tizen OS	5.0	Broadcom
Fossil	Explorist HR	Wear OS	4.2	Qualcomm
Apple	Watch 4	Watch OS 5	5.0	Apple
Huawei	Watch 2	Wear OS 2	4.1	Broadcom
Fitbit	Versa 2	Fitbit OS 4	5.0	Cypress S.
Sony	MDR-XB9	—	4.1	Qualcomm
Apple	AirPods 2	—	5.0	Apple

Table 2.2 – Bluetooth LE devices.

Vendor	Model	BT	Chipset
Apple	Watch 4	5.0	Apple
Fitbit	Charge 2	4.1	Microelectronics
Fitbit	Charge 3	5.0	Cypress S.
Huawei	Band 3e	4.2	RivieraWaves
Mi	Band 2	4.1	Dialog S.
Mi	Band 3	4.1	Dialog S.
Mi	Band 4	5.0	Dialog S.

Table 2.3 – Applications and Actions captured in the dataset. Some actions are application-specific, *e.g.*, `DiabetesM_AddCalorie`. Other actions, *e.g.*, `Workout`, exist in different apps and in the firmware of wearable devices.

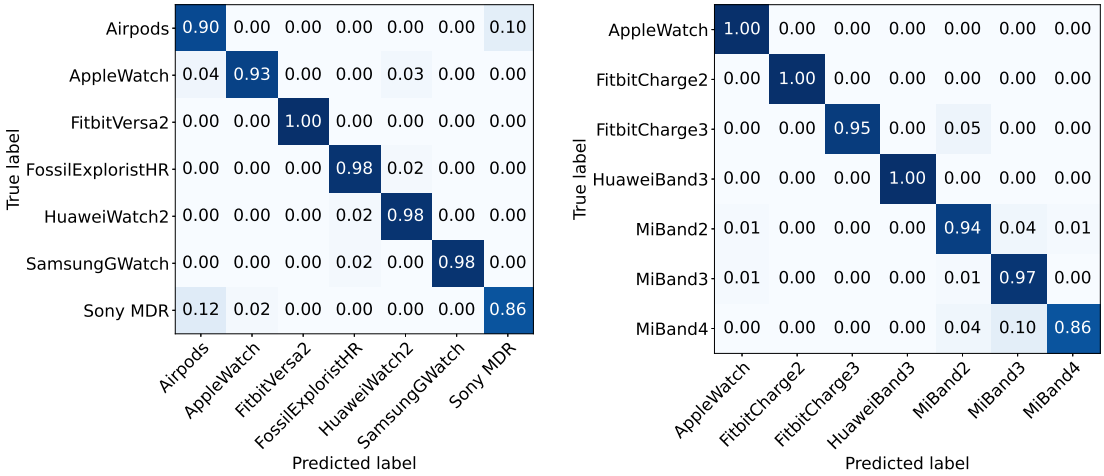
Applications	20Min, ASB, Alarm, AppInTheAir, AthanPro, AthkarOfPrayer, Battery, BeurerApp, Bild, Bring, Calm, Camera, ChinaDaily, Citymapper, DCLMRadio, DailyTracking, DenverApp, DiabetesM, DuaKhatqmAlQuran, Endomondo, FITIVApp, FITIVPlus, FindMyPhone, Fit, FitBreathe, FitWorkout, Fitbit, Flashlight, FoursquareCityGuide, Glide, GooglePay, GooglePlayMusic, HealthyRecipes, HeartRate, HuaweiApp, Kaia, KeepNotes, Krone, Lifesum, MapMyFitness, MapMyRun, Maps, Medisafe, Meduza, MiApp, Mobills, Music, MyFitnessPalApp, NYT, NoApp, Outlook, PearApp, Phone, PhotoApp, PillReminder, PlayMusic, PlayStore, Qardio, RamadanTime, Reminders, Running, SalatTime, SamsungHealthApp, Shazam, Sleep, SleepTracking, SmartZmanim, SmokingLog, Spotify, Strava, Telegram, Timer, Translate, Walgreens, WashPost, WearCasts, Weather, Workout.
Actions	AddCalorie, AddCarbs, AddFat, AddFood, AddGlucose, AddInsulin, AddProteins, AddWater, Browse, BrowseMap, CaloriesAdd, Coffees, EmailReceived, HeartRate, Leisure, LiveStream, NightLife, Open, PhoneCallMissed, PhotoTransfer, Play, Restaurants, Running, SearchRecipe, Shopping, Skip, SMSReceived, Sync, Walking, Workout.

2.5 Device Identification

We first focus on the adversarial goal G1: recognizing a device name/brand from the metadata of its encrypted Bluetooth traffic. This attack is a stepping-stone for other attacks that aim to infer more fine-grained information such as actions performed by the wearer or applications installed on her device (§2.6). We recall our assumption that the adversary A_{comm} does not observe the pairing event between a wearable w and the smartphone S, which would give him the device information in plaintext. Instead, we demonstrate how A_{comm} can infer this information from encrypted communication patterns. Many recent related works already highlight that current BLE devices do not rotate their MAC addresses sufficiently or at appropriate times, enabling tracking [28, 44, 154]. We highlight a deeper problem: the communication patterns (*e.g.*, inter-packet timings) are sufficient to accurately identify devices.

Attack. We use the methodology described in §2.4 and train two classifiers: one for identifying Bluetooth Classic devices and one for distinguishing BLE devices from their encrypted traffic. We use our captured dataset of encrypted traces with the `device` label as the classifier's target. Initially, we have 10,371 feature vectors across diverse applications/actions of 7 Bluetooth Classic devices, and 329 feature vectors for 7 Bluetooth Low Energy devices. The `AppleWatch` is in both categories as it uses both Bluetooth flavors.

The large sample difference in Bluetooth Classic is due to the automation of two Wear OS smartwatches (§2.6.2). As the Bluetooth Classic dataset is imbalanced with respect to the number of samples per device, we balance the samples per device label and maintain an



(a) Classic devices.

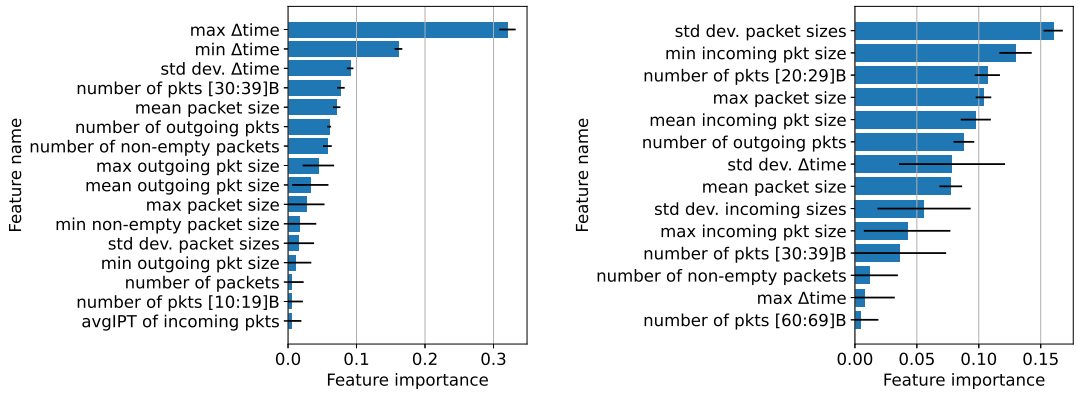
(b) LE devices.

Figure 2.4 – Normalized confusion matrix per true label for device identification. Values in the diagonal are the recall per class.

equal representation of each device's actions. This gives us between 20 and 60 samples per device, except for the `HuaweiWatch` that has 125 samples (1 per class). In total, the equalized Bluetooth Classic dataset consists of 326 feature vectors. We use a Random Forest classifier with 10 trees (our experiments showed that additional trees were not necessary – Appendix A.4, Figure A.1a) that we train using the features described in §2.4, and we do not limit their depth. We apply Recursive Feature Elimination and keep the most significant 10 features.

Results. For the multi-class classification problems with 7 Classic and 7 LE devices, the classifier's precision/recall/F1-score is 0.96 for Bluetooth Classic and 0.97 for Low Energy (Appendix A.5, Tables A.1 and A.2), thus showing that identifying a wearable device from its encrypted traffic is successful. We also find that our classifier accurately distinguishes between different models from the same vendor (*i.e.*, Mi Band 2, 3 or 4, and Fitbit Charge 2 or 3) indicating that each model has distinctive traffic patterns. Figures 2.4a and 2.4b show the confusion matrices. The values in the diagonals are the recall per class.

We perform a feature importance analysis and find that timings are crucial for discriminating among the Bluetooth Classic devices: all three most important features are related to inter-packet timings (Figure 2.5a). This corroborates the findings of Aksu *et al.* [3] who show that the traffic from 6 smartwatches has a distinct inter-packet timing distribution, and is in agreement with some fingerprinting results in other domains (*e.g.*, website fingerprinting based on Tor traffic [173]). In the case of Bluetooth Low Energy, the classifier selects primarily size-based features (Figure 2.5b). We postulate that this difference is due to the increased capabilities of Bluetooth Classic devices (*i.e.*, high-end smartwatches) compared to LE devices that consist of simpler devices. Smartwatches support a wide range of possible actions that can generate small or large amounts of data, which makes global-volume-based features less stable than timings that are inherently tied to the OS and the hardware. On the contrary, LE devices support only limited functionalities (*e.g.*, activity tracking or heart rate monitoring), and their communication pattern is inherent to the nature of the application, making size-based features discriminative across devices. Another possible explanation is that due to the absence of a block cipher on the link-layer, BLE packet sizes reveal more information to an eavesdropper. Finally, we observe that the relative feature importance is less pronounced than in the case



(a) Classic devices.

(b) LE devices.

Figure 2.5 – Feature Importance for device identification. `Δtime` is the sequence of inter-arrival times. `avgIPT_X` are the features from Saltaformaggio *et al.* [186] also describing inter-arrival times. The x-axis indicates the relative feature importance.

of Bluetooth classic, thus indicating that the classifier needs more features to successfully distinguish the samples.

Chipset Fingerprinting. Our classifier described above fingerprints a combination of the hardware, the firmware, the OS, and the applications installed. In Appendix A.1, we briefly explore if the chipset (*e.g.*, `Broadcom`) can be recognized from the encrypted communication. We observe that there exist stable communication patterns across the devices sharing the same chipset. However, the low number of devices (*i.e.*, 1 – 3 devices per chipset manufacturer) prevents from reaching a definitive conclusion.

Take-Aways. Our experiments confirm that there exist discriminating features across the encrypted traffic of Bluetooth wearable devices; an eavesdropper can use these features to differentiate devices. This raises a question about the level of protection offered by tracking countermeasures, *e.g.*, MAC address randomization, employed by the Bluetooth LE protocol. Moreover, our results show that Bluetooth Classic devices are distinguishable due to their communication time patterns, whereas LE ones have distinct size patterns. We also find that the traffic of devices with the same chipset manufacturer have common patterns. Despite these common patterns, devices from the same vendor can still be distinguished from each other using the device-identification methodology, demonstrating the robustness of our approach. Finally, we remark that device identification is an entry-level attack to other attacks (*e.g.*, action or application identification described next) that aim at inferring more sensitive information about the owners of wearable devices.

2.6 Action and Application Identification

We now consider the adversarial goals G2/G3 and focus on recognizing user-actions from their corresponding encrypted Bluetooth communications. Actions are related to the capabilities of wearable devices: measuring a heartbeat, beginning a workout, tracking a meal or medicine intake, playing music, etc. Our dataset analysis shows that most user-actions — even the mere opening of an application on a smartwatch — result in Bluetooth communication with the paired phone. In this section, we train a classifier to recognize user-actions from the metadata of these encrypted Bluetooth communications.

Due to technical constraints, our experimental evaluation is two-fold and "T-shaped":

1. **"Wide"-part** (§2.6.1): We run the attack on all the wearable devices of our testbed (Tables 2.1 and 2.2). Since most of them are not automatable due to their proprietary OS-/firmware, we use the samples that we manually trigger by performing the appropriate action (*e.g.,* performing a short running workout).
2. **"Deep"-part** (§2.6.2): We build and use an automation pipeline for Wear OS devices to generate more data. We use this enhanced dataset (1) to identify applications running on a smartwatch, for instance religious or medical applications, (2) to identify fine-grained actions within specific applications, for instance the action "record an insulin injection" in an application that is used to manage diabetes, and (3) to model an attacker capturing traffic over a longer period of time (hours) and attempting to recognize actions and applications in the trace.

Methodology. For our two-fold experimental evaluation, we follow the methodology described in §2.4 except that we enhance the set of extracted features (see next point). We use the `Action` (or `Application`) label as our classifier's target. In subsections §2.6.1 and §2.6.2, we include further details that depend on the specific experimental settings.

Feature Extraction & Classification. We slightly modify the features described in §2.4. First, we replicate the 3 packet sequences: (1) from Master to Slave, (2) from Slave to Master, and (3) the sequence consisting of all non-null, non-ACK packets. Then, we remove small packets from the copies: we empirically observe that small packets are not useful to the classifier because they are common across applications and actions. After experimenting with different thresholds, we filter out packets smaller than 46B from the 3 new time-series. As before, we extract the min/mean/max/count/standard deviation from these time-series, which leads to 15 additional scalar features. Furthermore, we tweak the features proposed by Liberatore and Levine [148] regarding the counts of packets in certain size ranges. Recall that, for device identification (§2.5), we used coarse, 10-byte wide buckets. However, our analysis of the Bluetooth traffic concerning user actions shows that some of them produce consistent and unique packet sizes. As a result, we define more fine-grained buckets and record the number of packets with size x, for $x \in [46; 1,005]$ bytes. We ignore packets above 1,005 B, the maximum payload size in our dataset. This leads to 960 scalar features replacing the 10 described in §2.4. Finally, we retain the same timing features as those described in §2.4. Overall, with our modified approach, we extract 997 features that we feed to our random forest classifier. To cope with the increased number of labels (*i.e.,* 49 actions in the "wide' experiment of §2.6.1 and 56 applications for the "deep" experiment of §2.6.2), we set the number of trees in the Random Forest algorithm to 30 and we use Recursive Feature Elimination to retain the 50 most important features (Appendix A.4, Figure A.1b justifies the choices of these values).

2.6.1 "Wide" Experiment on All Wearable Devices of our Testbed

We first focus on devices whose Bluetooth traffic capture can not be automated: step counters and fitness trackers (`Fitbit Charge 2-3, Huawei Band 3e, Mi Band 2-3-4`), and smartwatches (`Fitbit Versa 2, Apple Watch, Samsung Galaxy Watch`), all with proprietary OS/firmware. We also include manually-triggered actions from Wear OS watches (`Huawei Watch 2, Fossil Explorist`), but not machine-automated ones that we use in §2.6.2. The subset of the dataset that we use in this section consists solely of Bluetooth traces triggered by the corresponding human action. We demonstrate that most actions performed on the wearable devices under examination result in a distinctive encrypted Bluetooth com-

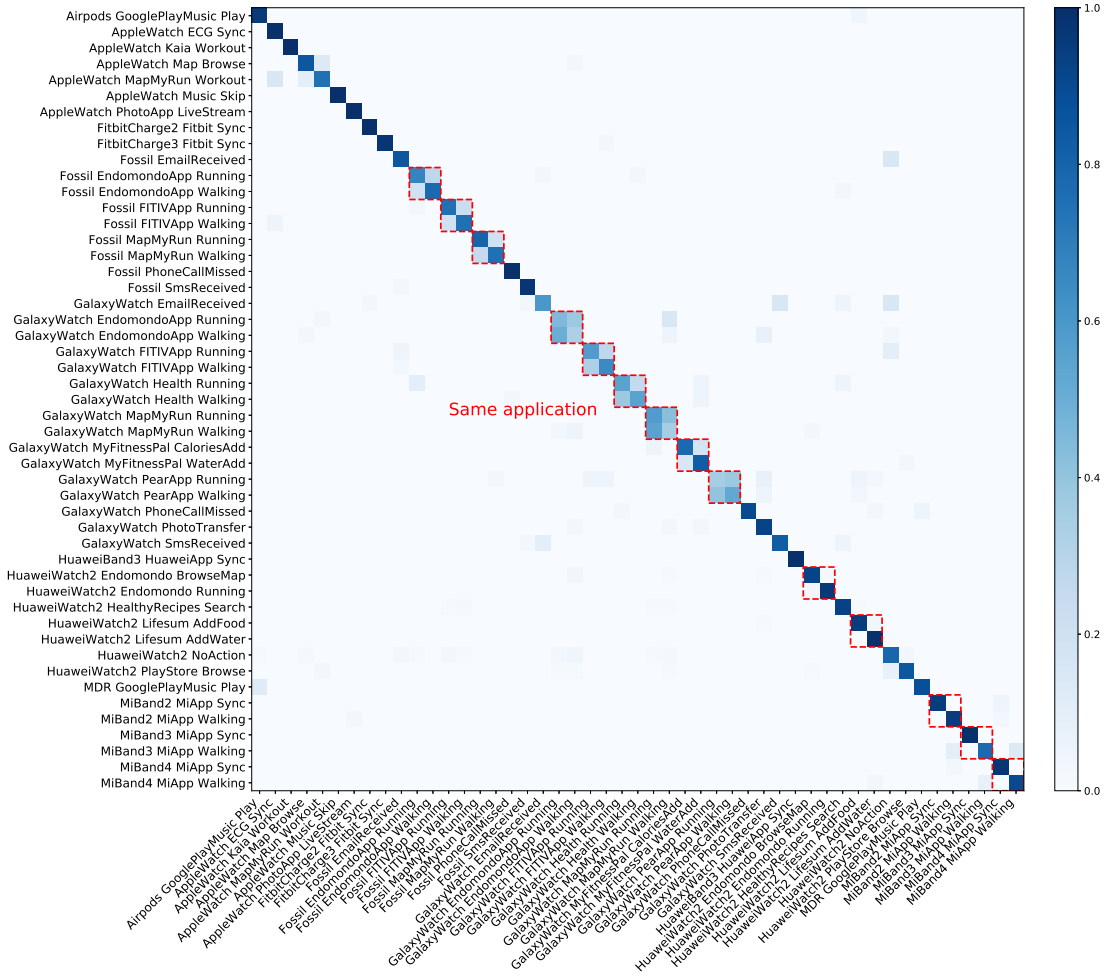


Figure 2.6 – Normalized confusion matrix per true label, Application and Action Identification ("wide" experiment). Red squares regroup actions within one application.

munication, and that an eavesdropper can automatically and passively recognize user actions (*e.g.*, starting a workout) and various events (*e.g.*, the reception of an e-mail/SMS/phone call).

Attack. We do not assume that the adversary knows the device; we train a classifier to infer both the device and the action in one step. A true positive occurs when both the device and the action are guessed correctly. We enumerate and collect 49 actions from the set of wearable devices considered and the companion apps installed on the phone (*e.g.*, `Measure Heartbeat`, `Record Food/Water` intake). The dataset is balanced: for each action, our dataset contains between 20 and 25 samples.

Results. The classifier performance over 49 actions is 82% precision/recall/F1-score (Figure 2.6 and Appendix A.5, Table A.7). We observe that most actions are recognized with a recall close to one which demonstrates that their corresponding Bluetooth communications have distinct patterns. This includes potentially sensitive user actions such as measuring heart rate, beginning a workout, receiving an e-mail or phone call. Moreover, there exist few classes with lower accuracy compared to others (with precision/recall between 40% and 60%). Our analysis shows that these classes concern actions within the same application, *e.g.*, `EndomondoApp_Running` or `EndomondoApp_Walking`, on the `SamsungGalaxyWatch`. However, this can be a limitation due to the number of samples (20–25 per class); we show in further experiments that fine-grained actions within one application can be inferred with

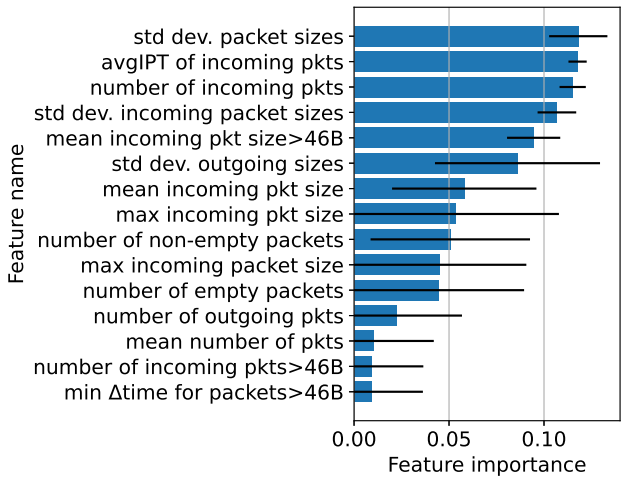


Figure 2.7 – Feature importance, Application and Action Identification ("wide" experiment).

more data (§2.6.2.3). We also observe that the classifier accurately labels the same action `Running` on devices from the same line of products: `MiBand2`, `MiBand3`, `MiBand4` demonstrating how it performs device and action identification in one step. Finally, we observe that our classifier yields the same accuracy for the devices `Fitbit2` and `Fitbit3` — which only `Sync` with their connected smartphone — thus confirming our Device Identification results (§2.5): an eavesdropper can recognize the communicating devices based on their metadata.

We perform a feature analysis and find that the most important features are based on communication volumes (`number of incoming pkts`, Figure 2.7) and packet sizes (features 1 and 4–8 of Figure 2.7), with a single highly-ranked feature about timings (`avgIPT of incoming packets`). This is consistent with our initial dataset analysis that demonstrated that various user actions trigger Bluetooth traffic with distinct packet sizes. Finally, we remark that in this experiment, the classifier recognizes the device and the application in one step. In §2.6.2.3, we improve the classifier's accuracy for identifying actions within the same-application by assuming that device and application identification has already taken place and by tailoring the training of our classifier to one specific application.

2.6.2 "Deep" Experiment on Wear OS devices

We now perform an in-depth analysis of Bluetooth communications' metadata on Wear OS smartwatches. We use automation to increase the size of our dataset, and we demonstrate the performance of our methodology on various adversarial tasks. To ensure that our synthetic (*i.e.,* computer triggered) actions generate plausible Bluetooth traffic, we restrict ourselves to a one-click action: the opening of an application on the smartwatch. We argue that this should be independent of the wearer's data and inputs and should generalize across users. The first purpose of this section is to demonstrate that the simple opening of a smartwatch application generates (encrypted) Bluetooth traffic patterns that can be accurately recognized by an eavesdropper (§2.6.2.1). Then, we investigate if the classifier trained by an adversary generalizes and transfers across different smartwatch-smartphone pairs, *i.e.,* to a different setup than that used offline by the adversary (§2.6.2.2). Furthermore, we build upon our device and application identification attacks, and we target a specific application used to manage diabetes to infer actions within that application. We also demonstrate how an eavesdropper can passively recognize sensitive, health-related actions (*e.g.,* record an insulin injection)

within this specific application (§2.6.2.3). Finally, we show how a persistent adversary that continuously monitors the Bluetooth traffic of a user can extract her profile by inferring her application openings and actions over the course of a day (§2.6.2.4).

2.6.2.1 Application Identification

We train our classifier to infer the opening of specific applications on a smartwatch. Identifying app openings has the benefit of being (1) independent of user actions and data more than of the recognition of actions (§2.6.1), and (2) easier to automate (and hence to train a classifier upon) for an adversary.

We remark that the mere presence of an app can leak sensitive information about the wearer. For instance, medical and well-being applications (*e.g.,* medication reminders, applications to stop smoking) hint that the wearer is concerned with a medical condition, and religious (*e.g.,* prayer time reminders) or news applications with a political orientation can reveal information about the users' beliefs. Even less revealing apps can be useful to a long-term adversary; users naturally install applications based on their interests and behaviors, and the list of apps on their wearable device can be exploited to build personal profiles. We argue that it is difficult to foresee whether the presence of an application is sensitive or not, especially when considering long-term profile building based on data from multiple sources, *e.g.,* for machine-learning-based advertising [71, 167]. We envision that as Bluetooth sniffing technologies are becoming less expensive (see our discussion in §2.8.1), Bluetooth traffic could become a valuable source of information. Companies are currently experimenting with Bluetooth-based "proximity advertising", a technology used to track users and display local targeted advertisements in transportation systems, airports, and supermarkets [7, 135, 196].

Automation Pipeline & Applications. We use the automation part of our Bluetooth traffic capture pipeline (§2.4) that consists of a Linux laptop that coordinates with a Windows machine that records traces via a Bluetooth sniffer. Using adb and monkeyrunner [104], the Linux laptop issues synthetic clicks and swipes to a Wear OS smartwatch connected over Wi-Fi. We force the watch to send data over Bluetooth (rather than Wi-Fi) by making sure that the Wi-Fi network does not have Internet access. We do not perform UI fuzzing; we manually specify the clicks and swipes needed to perform the desired actions on the watch. The exact same clicks and swipes are reused to repeat an action. At the time of writing, due to the lack of debugging tools, only Wear OS smartwatches could be automated in the desired way. Devices running Tizen OS should support automation using a variant of adb called sdb [213], however, the current API does not enable it. Our first experiment uses a Huawei Watch 2 (LEO-BX9) running Wear OS 2.16, paired with a Pixel 2 running Android 9.

We select 56 applications from the Google (Wear OS) Play store, favoring top-rated applications per category. Our choice of applications was constrained by the availabilities of apps on the Swiss Play store. Our list includes:

- Religious apps (`DuaKhatqmAlQuran`, `AthkarOfPrayer`, `SalatTime`, `DCLMRadio`)
- Health-related apps (`DiabetesM`, `Medisafe`, `SmokingLog`, `Qardio`, `HeartRate`)
- Lifestyle-related apps (`Lifesum`, `Calm`, `DailyTracking`, `HealthyRecipes`, `SleepTracking`, etc)
- Sport/Activity-related apps (`Endomondo`, `FitWorkout`, `FITIVPlus`, etc)
- Local newspapers (`ChinaDaily`, `WashingtonPost`, `Meduza`, `Krone`)
- Mobile banking and finances (`ABS`, `Mobills`)
- "Local guides"/maps (`Citymapper`, `Foursquare`)
- Communication apps (`Telegram`, `Glide`, `Outlook`)

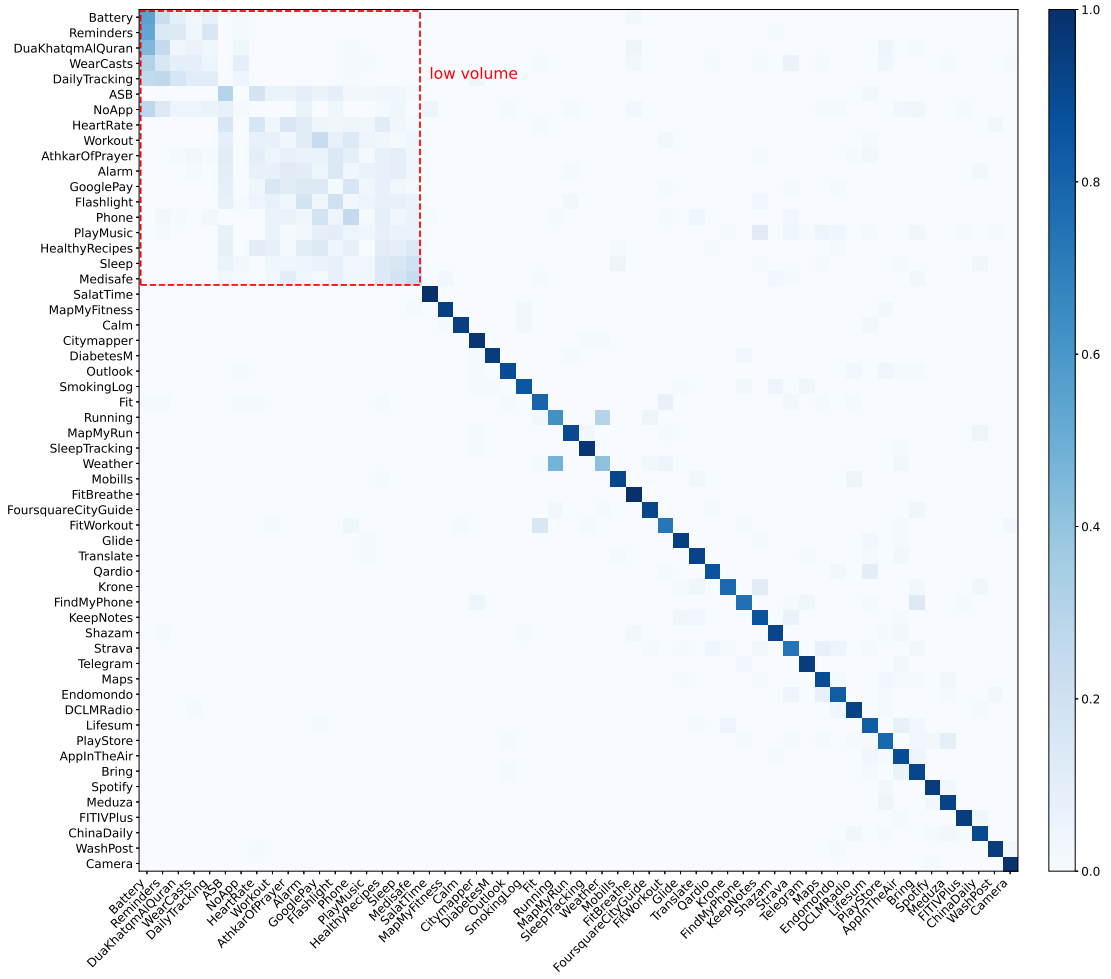


Figure 2.8 – Normalized confusion matrices per true label for recognizing smartwatch application openings. The applications are sorted by increasing median transmitted volume.

We also include stock applications (*e.g.*, `Reminders`, `Weather`, `Phone`), as well as common applications (*e.g.*, `Telegram`, `Translate`) as a control for the sensitive groups and to increase the number of applications. Finally, we include a `NoApp` label that corresponds to background communications between the smartwatch and the phone.

Results. The classifier's performance (precision/recall/F1-score) over the 56 apps is 64%. However, we find that the precision and recall per class varies greatly. In particular, the majority of apps (39 out of 56 apps) is classified with a mean accuracy close to 1, whereas a smaller subset of the apps is confused among each other by the classifier. Our analysis shows that this concerns apps that trigger minimal Bluetooth communications upon their opening (*e.g.*, `Battery`, `Flashlight`); this fact is visible when we order the confusion matrix by the median transmitted volume (Figure 2.8). We call these apps "low-volume", as they communicate less than 200 bytes for at least 75% of their samples. However, except for `Battery` and `Reminders`, which are the unique apps that trigger absolutely no communication, we find that low-volume apps communicate a small amount of information on the Bluetooth layer (Appendix A.5, Table A.8). To further investigate the difference across the two subsets of apps, we train two separate classifiers: one specifically targeting the 18 low-volume apps and another one to distinguish among the remaining 38 non-low-volume apps. The mean accuracy of the first classifier reaches 17% (Appendix A.5, Table A.10) with high variance across the labels,

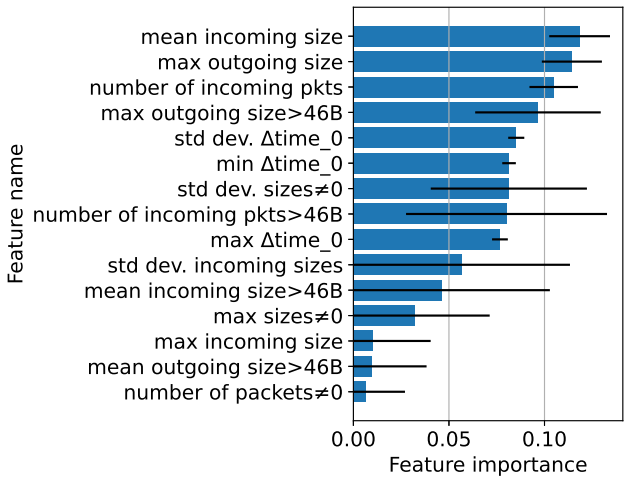


Figure 2.9 – Feature importance for application identification, “deep” experiment.

which shows that the data exchanged by these apps is simply too small and/or too variable to be learned by the model. We find that the `NoApp` label that corresponds to background communications between the smartwatch and the smartphone is part of the low-volume group. This indicates that low-volume apps are not only hard to distinguish among their peers but are also difficult to differentiate from the absence of activity. On the contrary, the non-low-volume apps are recognized by the second classifier with a high accuracy of 90% (Appendix A.5, Table A.11), demonstrating the practicality of the attack in this case. We observe that — as before — the important features are based on the sizes of packets and the timings (Figure 2.9).

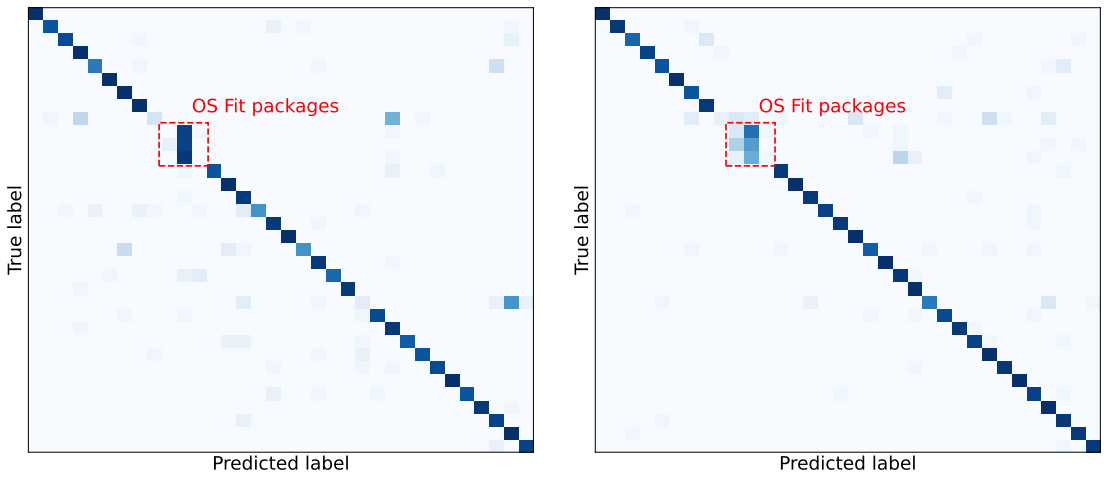
2.6.2.2 Model Transferability & Aging

Our application-identification attack was successful on a single smartwatch-smartphone pair (§2.6.2). However, this attack requires the adversary to train a classifier on the particular pair of devices that the target possesses. In this subsection, we investigate if the trained model generalizes to other devices which the adversary not used for training.

In more detail, in the previous experiment we use a Huawei Watch 2 (LEO-BX9) running Wear OS 2.16, paired with a Pixel 2 phone running Android 9. In this experiment, we include a new pair of devices: a Fossil Q Explorist HR smartwatch running Wear OS 2.16, paired with a Nexus 5 phone running Android 6. We could not downgrade one Wear OS device to a different version to introduce more variability to our experiment. Also, we could not include the Apple Watch / iPhone pair in the transfer experiment due to a lack of overlap in the apps that we selected and the apps available in the Apple Store. Hence, we leave the question of cross-OS transferability as an interesting future work.

We select 34 apps from the Huawei-Pixel pair that could also be installed on the Fossil-Nexus pair (Appendix A.5, Table A.5). We follow the same methodology, except that we use all the samples collected from one pair of devices as the classifier’s training set and the data collected from the other pair as the testing set. We perform our experiments in both directions.

Our results show that the trained model generalizes well: it has a precision/recall/F1-score of 81% when the data collected from the Huawei-Pixel pair is part of the training set and the data collected from the Fossil-Nexus pair is the testing set (Figure 2.10a, and Appendix A.5,



(a) Train on Huawei-Pixel. Test on Fossil-Nexus.

(b) Train on Fossil-Nexus. Test on Huawei-Pixel.

Figure 2.10 – Transferability experiment for the application identification ("deep" experiment).

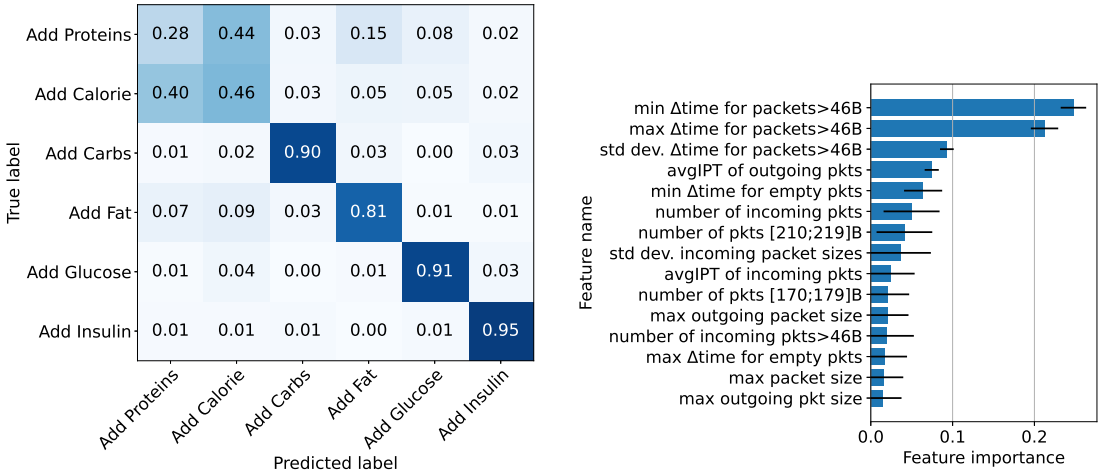
Table A.12). The classifier's precision/recall/F1-score reaches 86% when we perform the experiment in the opposite direction (Figure 2.10b, and Appendix A.5, Table A.13). Some apps are misclassified: our analysis shows that these are applications that are native to the OS (`Fit Packages`). This is not surprising, given the differences in major Android versions. However, and more importantly, our experiment shows that fingerprinting non-native applications by their (encrypted) Bluetooth traffic is possible independently of the smartphone's OS version, which demonstrates the robustness of our attack methodology.

Dataset Aging. In Appendix A.2, we briefly explore the effect of dataset aging, that is, the loss of accuracy incurred as the adversary uses older datasets. We observe that although performance vary over a month, it remains generally high (Figure A.1). We observe that some applications with dynamic traffic have naturally the lowest F1-score (Figure A.2a), but that training over a few days may help improve the performance over the whole month (Figure A.2b).

2.6.2.3 Fingerprinting Fine-grained Actions Within an Application

We now use the application-opening identification (§2.6.2) as a stepping stone to another attack that aims at inferring potentially sensitive actions within one particular application. For a use-case, we choose the app `DiabetesM` that helps people diagnosed with diabetes to keep track of their meals and medicine intakes. Although the Huawei smartwatch that we use for this experiment is not marketed as a medical device, this information is of medical nature.

We follow the previous methodology and use the automation tool to generate traffic that corresponds to the usage of the `DiabetesM` app. We manually program user interactions (*i.e.,* pressing buttons) within the application and capture the traffic of 6 actions related to the management of meals and medicine intakes: `Add Calorie`, `Add Carbs`, `Add Fat`, `Add Glucose`, `Add Insulin`, and `Add Proteins`. We collect 150 samples per action, map them to feature vectors using the features described in §2.6 and split them into 80% for training and 20% testing. Then, we train a classifier tailored to this particular app that aims to distinguish among the 6 possible user actions, *i.e.,* we assume that the application's traffic has been already classified as `DiabetesM`.



(a) Normalized confusion matrix per true label.

(b) Feature importance.

Figure 2.11 – Fingerprinting fine-grained action within on application: `DiabetesM`.

The overall accuracy of the classifier over the 6 actions is 70% (Figure 2.11a, and Appendix A.5, Table A.4), which is significantly better than guessing at random. This indicates that actions within a specific application generate distinct Bluetooth traffic that can be fingerprinted by an eavesdropper. More importantly, our results show that the sensitive action `Add Insulin` is recognized with a precision/recall/F1-score of 0.9/0.95/0.92%, respectively, *i.e.*, the encrypted communication patterns generated by this sensitive action "stand out" from other actions of the `DiabetesM` app.

We here remark that all of these actions are semantically similar: they all update a variable in a database stored on the paired smartphone. We expect that developers could prevent the traffic-analysis attacks by simply padding the traffic corresponding to all of these actions to a constant size. However, the feature analysis reveals that *timings* matter most, not sizes (Figure 2.11b). A closer inspection reveals that a human has to interact with `DiabetesM` in two consecutive steps: (a) by increasing a value (*e.g.*, pressing "record insulin injection"), and, (b) by clicking on a "save" button on a different screen. To our surprise, both actions generate traffic, and the classifier detects the timing differences between the two actions.

First, this highlights a limitation of our experiment. In our methodology, the duration between the press of the first button, the swipes, and then the press of the "save" button are constant and precisely reproduced by the automation framework. A human would produce more variable durations that would be harder to fingerprint by their encrypted traffic. However, we believe that this finding is still an interesting showcase for the capabilities of traffic-analysis attacks: padding each action into a similar message is not sufficient, because the classifier can "count" the number of swipes from the screen containing the save button back to the screen containing the target action. Hence, it is necessary to obfuscate this duration, or rethink the strategy that the application employs to synchronize data with the smartphone.

2.6.2.4 A Persistent Adversary

We now consider a longer-term adversary that aims at identifying the actions performed on a smartwatch by its wearer over the course of a day. This adversary could be a nosy neighbor or an office eavesdropper, capturing Bluetooth traffic continuously over a long period, and attempting to monitor the habits of the target. This experiment drops a simplifying assump-

tion made in the previous experiments, *i.e.,* that the adversary knew when the traffic related to an action starts and stops. Indeed, all previous experiments used 30-second traffic samples, each corresponding to exactly one action. In this new experiment, the adversary records a continuous traffic capture over 24 hours, and its goal is to output a series of predictions over this period. Furthermore, the adversarial task is slightly tweaked: the prediction can be either a user-action, or the `NoAction` label corresponding to the absence of user activity. Finally, due to their long duration, these traffic captures contain background traffic (updates, synchronization) more than the short 30-sec samples used previously. The goal of the adversary is therefore to distinguish specific app openings from background noise and OS communications.

Methodology. We generate application openings and user actions with the `HuaweiWatch2` smartwatch, using the automation framework presented in the "deep" experiment (§2.6.2). We simulate one user who wears the watch for 24 hours. Over the course of a day, we model the user's interactions with her smartwatch following a recent user study that quantifies smartwatch usage in the wild [149]. In particular, for each 1-hour slot, the number of interactions is drawn from a probability distribution favoring daytime hours over nighttime ones (Figure 4 of [149], page 389). User actions are not triggered with an equal probability: popular applications such as messaging/e-mails, maps, alarms/clocks, and fitness trackers, are more likely to be triggered than others (Table 6 of [149], page 390). We model this by updating their prior probability to 2× compared to that of non-popular applications. We select 33 high-volume applications from popular categories and enumerate 17 user actions within these applications, *e.g.,* `DiabetesM_AddInsulin`, `HealthyRecipes_SearchRecipe`, `FitWorkout_Open` (Appendix A.5, Table A.6). Individually, each of these actions has a short duration (≤ 20sec). However, these fine-grained actions follow each others in semantic sequences: *e.g.,* we automate the sequence `Endomondo_Open`, `Endomondo_Running`, waiting 2min, `Endomondo_Close`, which is the equivalent of a 2-min workout. The classifier attempts to recognize each individual action (except for `_Close` actions).

Then, we automate the recording and triggering of actions. Due to technical constraints, we record 20min-captures that we concatenate to form a 24h capture. The parameters of each 20min-capture are drawn from the distribution of the modeled time of the day. Within one capture, the simulation is a simple state machine that loops over the following actions: (1) it flips a biased coin deciding whether to trigger an action or not, and (2) if the outcome is positive, it draws one action at random following the biased probability distribution, runs the action, and waits for a random time defined by the expected number of events in the modeled time of the day. In parallel, the smartwatch and smartphone normally exchange background data.

Attack. To train its classifier, the attacker uses the short 30-sec captures corresponding to the 50 classes (*i.e.,* applications and actions) selected. Moreover, it employs a 51st class that it uses to model noise: this class contains the background communications of the wearable, recorded as `NoApp_NoLabel`. In addition, we notice that closing an application also generates network communications. The volume exchanged is low, hence they are difficult to classify. However, we observe that when treated as noise, they are useful in helping the classifier distinguish between actions of interest and background traffic. Therefore, we add the classes `AppX_Close` for all 33 applications of interest. The adversary's dataset is balanced: it contains between 30 and 40 samples per class. Finally, the attacker extracts features, as in the previous experiments (§2.6.2), and trains a Random Forest classifier.

During testing, the attacker is provided with the uncut 24-hour traffic capture. First, it runs a splitting algorithm that identifies sequences in the capture that possibly correspond to

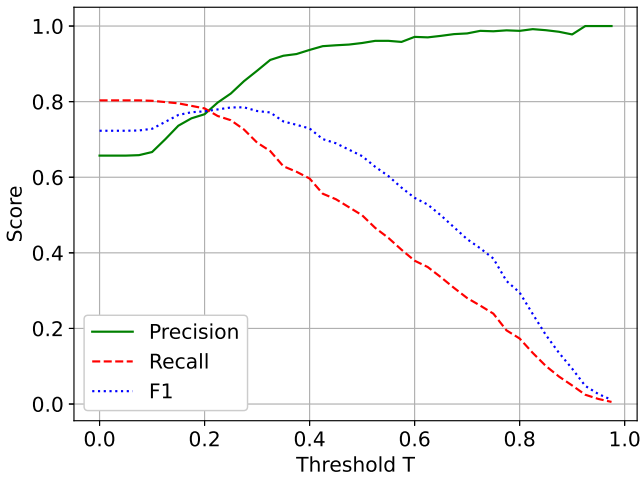


Figure 2.12 – Attacker's score when classifying events over 24-hour captures. The threshold is the minimum confidence needed to output a prediction.

user-actions. This splitting algorithm uses a sliding window and records the times at which the sum of bytes exchanged in the window is greater than 200 bytes, following the criteria for "high-volume" apps presented in §2.6.2. Then, it classifies the contents of each window; however, it only outputs the most likely class if its probability is greater than a confidence threshold T. If no class meets this criteria, it outputs `NoAction`.

Evaluation Metrics. We adapt the attack's evaluation metrics to the new task. A true positive corresponds to a correct prediction in the "correct" time-interval, *i.e.,* if the time intervals of the real action and the predicted one overlap. A false positive occurs when the attacker's classifier outputs a label other than `NoAction` that does not overlap with a real action of the same label. Finally, a false negative is when the classifier misses a real action. Following these definitions, we calculate the classifier's precision/recall/F1-score as usual.

Results. The results are computed over the $72 \times 20\text{min} = 24$ hours of the experiment. We parameterize them with the confidence threshold T, which impacts the overall sensitivity: lower T values result in a higher recall and lower precision, and vice-versa. For the classification task with 51 classes, the classifier's average precision ranges from 0.65 to 1, and its mean recall per class from 0.7 down to 0 (Figure 2.12). The maximum recall is 83.5%, for a threshold T of 0.1, and the corresponding precision is 74.9%. The maximum precision is 1.0%, for $T = 0.6$, and the corresponding recall is 23.5%. The best F1-score is 0.83 and is achieved at $T = 0.25$. This experiment demonstrates that a persistent adversary successfully recognizes high-volume applications from the absence of activity and accurately classifies them over the course of the day. The different values for the confidence threshold T indicate a precision/recall trade-off. An adversary can choose to optimize its strategy towards one metric or the other (or both) depending on its goals. For instance, an adversary who aims at recognizing, with high precision, an application of interest or a particularly sensitive action (*e.g.,* , `AddInsulin`) could do so at the cost of more false negatives (and lower recall). Whereas, another adversary, *e.g.,* a smart billboard displaying an advertisement to a passer-by, aiming to identify the set of applications and actions of its target (that can help to build a profile) could choose a lower decision threshold and achieve better overall performance.

2.6.3 Summary of the Attacks

Overall, the experimental results of this section demonstrate that different actions (*e.g.,* declaring an insulin shot on a diabetes monitoring application) performed on a wearable device trigger unique Bluetooth traffic. These communication patterns can be recognized by an eavesdropper (*e.g.,* a nosy neighbor or a proximity-based advertiser) to infer the action performed despite the encryption. This holds across the multiple wearable devices such as smartwatches and fitness trackers from diverse vendors. We also find that the mere opening of an app on a smartwatch generates distinguishable traffic, leaking potentially sensitive information that is associated with the presence of the app (*e.g.,* a religious or a political app). For application-openings, we identify a subset of applications that is inherently well-protected against traffic analysis: "low-volume" applications. This hints that minimizing data exchanges is an obvious natural defense against our attacks. Furthermore, our results demonstrate that our attacks generalize well across different devices: we show that a model trained for recognizing application openings on one smartwatch-smartphone pair can be applied with high accuracy on another smartwatch-smartphone pair. This suggests that our methodology can be cost-effective for an adversary. Finally, we demonstrated how a persistent long-term tracking adversary can benefit from our traffic-analysis attacks by employing them to profile users, *i.e.,* infer their habits and actions on their daily lives, and that dataset aging does not significantly affect the attack's performance.

2.7 Protections

In the previous sections (§2.5, §2.6), we demonstrated how an eavesdropper capturing Bluetooth communications between a wearable device and its connected smartphone can perform traffic-analysis attacks and infer information such as the device model, the applications installed on it, and the actions performed by the wearer. In this section, we investigate defense strategies against Bluetooth traffic-analysis attacks. We first review the existing types of defenses against traffic-analysis attacks and identify the most popular approaches. Then, we implement these strategies and evaluate them against our traffic-analysis attacks.

High-level Defense Strategies. Before diving into the design of a defense, we remark that data minimization is a simple, inexpensive, and valid approach: data that is not exchanged cannot be fingerprinted. Indeed, our "deep" experiment shows that applications with low traffic volumes are naturally better protected against traffic analysis (§2.6.2). In a similar vein, infrequent "bulky" communications (*e.g.,* syncing a step counter only once a day at midnight) make the adversary's task harder in two ways: by leaking less metadata about timings, and by requiring the adversary to observe the communication at the right time. However, such high-level strategies do not apply to all applications (*e.g.,* interactive applications such as newspapers, radios or music players).

Defense Design. The purpose of this section is not to discover a perfect defense: it might not exist or its cost might be unbounded. Rather, we evaluate the effectiveness of common protections against traffic analysis with respect to the attacks that we presented earlier and study the communication overheads that they introduce. Fundamentally, a classifier properly trained can detect even small differences between two network traces. Ensuring that all network traces are perfectly indistinguishable from each other is infeasible. Therefore, we analyze the effectiveness/cost trade-offs introduced by standard defenses and discuss their feasibility for the protection of Bluetooth communications.

Defense Categories. We first perform a brief taxonomy of defenses against traffic analysis. The most active research fields are focused on the Tor network [41, 82, 102, 126, 152, 164, 225, 228, 237] and on IoT traffic/smart-homes [6, 16, 17, 82]. In a different category, recent anonymous messaging protocols often resist traffic analysis against a much stronger global adversary, at the cost of having a high bandwidth [20, 50] or a high latency [10, 61, 137]; indeed, spending time or bandwidth is a fundamental trade-off for achieving traffic-analysis resistant communications [70]. To the best of our knowledge, traffic-analysis defenses have not yet been explored on wearable devices.

We distinguish three defense categories [102]: regularization, obfuscation, and randomization:

- **Regularization** defenses make packet traces harder to distinguish by removing their differences, *e.g.,* by enforcing constant bit-rates and packet sizes [17, 40, 41, 82], by altering the traces into the common closest "super-sequence" of packets [225], or by forbidding duplex communications [228].
- **Obfuscation** approaches aim at confusing the adversary by tweaking the setting, *e.g.,* by hiding traffic into another protocol [152, 237], or by loading two web pages at the same time, as in the case of website fingerprinting [164].
- **Randomization** defenses confuse the adversary by adding randomized dummy traffic [6, 16, 82, 102, 126].

Defenses based on regularization are often easier to reason about and to analyze their formal guarantees, but they have the downside of being more costly than the others. On the contrary, obfuscation approaches consist of more practical defenses that often assume a certain type of adversary, *e.g.,* that cannot de-multiplex encrypted web pages, or recognize Tor traffic hidden as Skype traffic. We do not explore obfuscation strategies as this category does not apply well to wearable devices that only support a few classes of traffic. One possible obfuscation strategy (not explored in this work) could be to split the traffic between Wi-Fi and Bluetooth, for smartwatches that are capable of both. Finally, randomization defenses tend to be the most lightweight. However, their efficacy evaluation is harder and is typically done using the success rate of the state-of-the-art attacks [102, 126].

Defense Evaluation. Our goal is to investigate and understand what degree of protection against Bluetooth traffic-analysis attacks would be provided by a practical and lightweight defense. We remind the reader that our classifiers use features based on timings (Figures 2.5a, 2.7, and 2.11b) and packet size distributions (Figures 2.5b and 2.7). Thus, we evaluate three orthogonal defenses that mask real sizes and timings, and that inject dummy packets (which achieves both):

1. `pad`: A lightweight regularization defense. Each Bluetooth packet is individually padded to a maximum size (255B for BLE packets and 1,021B[3] for Bluetooth Classic packets). Per-packet padding hides specific sizes and unlike per-flow padding, it incurs no delay (ignoring the small delay due to transmitting larger packets).
2. `delay_group`: A regularization defense that delays and groups packets to the next second. This obfuscates fine-grained timing information by imposing a pace. This approach is clearly incompatible with latency-sensitive Bluetooth communications such as audio streaming, real-time and interactive applications. It does not incur bandwidth overhead.
3. `add_dummies`: A randomization defense that injects packets at times drawn from a Rayleigh probability distribution. The use of the Rayleigh distribution is inspired by the

[3] This corresponds to the max payload of a 3-DH5 ACL packet in Bluetooth Classic.

"Front" part of Front/Glue [102], a state-of-the-art lightweight randomization defense designed for website fingerprinting. We experimentally select 6s for the mean of the Rayleigh distribution, and 300 for the number of dummies we generate (Appendix A.4, Figure A.2). Finally, we sample the size of each dummy from a distribution created with the collected samples. Therefore, this defense assumes that the defender knows a priori the distribution of packet sizes.

To assess the protection level provided by the defenses, we measure the accuracy achieved by the classifier trained by the adversary. We assume that the adversary knows the defense in use and is able to adapt the training of the classifier. To quantify the cost of each defense, we use 5 metrics: the mean delay introduced per packet, the total added duration to the sample, the number of bytes added (both in terms of padding and dummy messages), and the total size overhead in percentage.

2.7.1 Experimental Results

We apply the various defenses on the Bluetooth traffic traces used for the device identification attack (§2.5), the "wide" experiment consisting of human-triggered actions on all wearable devices (§2.6.1), the "deep" experiment consisting of automated apps openings on Huawei Watch 2 (§2.6.2), and the fine-grained action recognition within the `DiabetesM` application (§2.6.2.3). This enables us to evaluate the performance of the defenses against multiple adversarial goals and various traffic settings.

Tables 2.4, 2.5, 2.6, 2.7, and 2.8 display the performance and cost of each defense against the attacks considered. The accuracy of the attack is averaged over the possible classes, and the defense costs are averaged per traffic sample. Our first immediate observation is that regardless of the attack and the defense, the mean accuracy achieved by the adversary's classifier is still significantly better than random guessing, which indicates that the defenses are far from being "strong" ones that provide cryptographic guarantees. Overall, the cost of each defense lies between 1–23× in terms of data transmission (at most ≈400 kB of extra data).

Device Identification. Tables 2.4 and 2.5 show that all defenses yield, at best, a moderate drop in this attack's accuracy. However, both flavors of the device identification attack are performed on a small number of devices (7 both for Bluetooth Classic and Low Energy). It is therefore not surprising that hiding the traffic of a device into that of another is a difficult task for the defenses. Among the evaluated defenses, we observe that the `delay_group` is the most effective one: it diminishes the attack's accuracy by 29 percentage points in the case of Bluetooth Classic, and 17 for Bluetooth Low Energy. However, the cost of `delay_group` is prohibitively high (≈ 0.5s delay added per packet) for a defense that is meant to be applied to all the communications performed by a device. Moreover, we find that the defense `pad` is ineffective with both Bluetooth flavors. This is not surprising for the case of Bluetooth Classic where the attacker's classifier relies mostly on timing features (Figure 2.5a). For Low Energy, features that relied on packet sizes (Figure 2.5b) have been replaced by the same top-rated timing features as in Bluetooth classic (`max/min/std of` Δ`time`, Appendix A.4, Figure A.3a). This suggests that these three features are important for device identification, regardless of the Bluetooth flavor and corroborates the findings of Aksu *et al.* [3] on smartwatches. We observe that `add_dummies` is lightweight and reduces the attacker's accuracy by 18 and 12 percentage points for Bluetooth Classic and LE, respectively. However, its performance is not uniform across the classes (Appendix A.4, Figure A.4). The confusion matrix shows that `add_dummies` only moderately protects the 3 `Mi Band 2-3-4` devices and does not protect the others.

Table 2.4 – Analysis of the defenses against device identification, Bluetooth Classic devices.

Defense	Acc. [%]	Delay/pkt [s]	Extra dur. [s]	Pad. [kB]	Dummy [kB]	Overh. [%]
No defense	96.3	-	-	-	-	-
pad	93.8	-	-	401.6	-	203.2
delay_group	67.7	0.5	0.2	-	-	-
add_dummies	78.0	-	-	-	92.9	47.0

Table 2.5 – Analysis of the defenses against device identification, Bluetooth LE devices.

Defense	Acc. [%]	Delay/pkt [s]	Extra dur. [s]	Pad. [kB]	Dummy [kB]	Overh. [%]
No defense	97.7	-	-	-	-	-
pad	94.5	-	-	139.0	-	277.1
delay_group	80.6	0.5	0.1	-	-	-
add_dummies	85.8	-	-	-	20.4	40.6

Table 2.6 – Analysis of the defenses against action identification, “wide” experiment.

Defense	Acc. [%]	Delay/pkt [s]	Extra dur. [s]	Pad. [kB]	Dummy [kB]	Overh. [%]
No defense	82.3	-	-	-	-	-
pad	64.1	-	-	272.0	-	270.5
delay_group	52.0	0.5	0.2	-	-	-
add_dummies	64.0	-	-	-	62.5	62.2

Table 2.7 – Analysis of the defenses against application identification, “deep” experiment.

Defense	Acc. [%]	Delay/pkt [s]	Extra dur. [s]	Pad. [kB]	Dummy [kB]	Overh. [%]
No defense	64.4	-	-	-	-	-
pad	27.9	-	-	150.5	-	585.0
delay_group	37.3	0.5	0.4	-	-	-
add_dummies	33.8	-	-	-	46.8	182.0

Table 2.8 – Analysis of the defenses against action-identification in DiabetesM application.

Defense	Acc. [%]	Delay/pkt [s]	Extra dur. [s]	Pad. [kB]	Dummy [kB]	Overh. [%]
No defense	70.4	-	-	-	-	-
pad	61.1	-	-	77.8	-	2374.8
delay_group	60.1	0.5	0.1	-	-	-
add_dummies	61.7	-	-	-	11.8	360.2

“Wide”-Experiment. We evaluate the performance of the defenses against the action identification attack demonstrated in §2.6.1. Compared to the previous attack, the task is different, and the classifier has to account for more labels (*i.e.,* actions). We find that all defenses perform better in general, reducing the attacker’s accuracy between 18 and 30 percentage points (Table 2.6). In particular, the difference in efficiency between pad and delay_group is now of only 12 percentage points, for a cost of 272 kB per sample for pad, and 0.5 sec delay for delay_group. This result suggests that both masking individual sizes or masking fine-grained timing information can help; developers could select the appropriate defense based on the cost (either in bandwidth or latency) that best matches their requirements. Finally, add_dummies performs similarly to pad but with a lower cost (62.5 kB per sample versus 272 kB for pad). However, add_dummies requires that the distribution of packet sizes is a priori known to generate dummies of plausible size. It is unclear how to compute this distribution for a defense meant to be applied to multiple devices from different vendors. One option would be to combine add_dummies and pad to avoid this requirement (we experimented with it and observed better effectiveness at a higher cost). Finally, the protection provided by the defenses is not uniform (we provide an example with add-dummies in Appendix A.4, Figure A.5a, other

defenses yield similar results), but unlike in the "deep" experiment (§2.6.2), the precision/recall per class does not seem correlated with the transmitted size. Similarly, the cost of `padding` varies greatly with the classes: it has a mean of 272 kB added, but a median of only 96 kB and a standard deviation of 650 kB. The costs soar up to 4.3 MB with streaming applications such as `AppleWatch_PhotoApp_LiveStream` or `PhoneCallMissed`. The defenses `delay_group` and `add_dummies` are more consistent, with a standard deviation of, respectively, 0.4 sec of delay per packet and 7 kB of dummy traffic.

"Deep"-Experiment. The traffic in this experiment consists of automated app openings on a Wear OS smartwatch. For this type of traffic, we observe that all three defenses perform similarly, reducing the attacker's accuracy by 31–36 percentage points down to ≈ 30% mean accuracy (Table 2.7). This highlights that on a specific class of traffic, *i.e.*, with more homogeneous traffic, the defenses are more efficient. In this case, `add_dummies` is the least expensive defense, requiring 46.8 kB of dummy traffic/sample, which is in the range of what the heaviest applications naturally use (Appendix A.5, Table A.9). A deeper analysis of the results also shows that all defenses successfully confuse the attacker for "medium-volume" apps (Appendix A.4, Figure A.5b). "High-volume" applications still stand out, but the gradual hiding visible on Figure A.5b suggests that increasing the parameters of the defense, *e.g.*, injecting more dummies in `add_dummies`, could potentially protect better such applications. However, this protection would come at an even higher cost for the applications that transmit smaller amount of traffic. As in the "wide" experiment, we observe that `pad` has a highly variable cost ranging from 6 kB to 1.6 MB. We observe the latter on the opening of the `Camera`, which suggests that `pad` is not adapted for constant-traffic. As before, `delay_group` and `add_dummies` have a consistent cost across labels.

Fine-Grained Action Fingerprinting on `DiabetesM`. In this case, we observe that all 3 defenses `pad`, `delay_group`, `add_dummies` are only moderately effective, reducing the attacker's accuracy by ≈ 10 percentage points. One possible explanation is the increased number of samples (150/label) compared to the previous experiments (25/label) that enable the adversarial classifier to adapt better to the defenses. In §2.6.2.3, we highlighted that timings were of importance to classify fine-grained actions in the application `DiabetesM`. However, in this case the attacker fingerprints a combination of the sizes and the timings, as the feature importance on `delay_group` defended traces reveals (Figure A.3b). In this experiment, we observe that all three defenses have a consistent cost across labels.

2.7.2 Summary of the Defenses

Our experimental evaluation of defense approaches, such as regularization and randomization, against the traffic-analysis attacks presented in this work yields some interesting insights. First, we find that these defenses achieve only a limited protection against our traffic-analysis attacks: although they do reduce the attacks' accuracy, the classifier trained by an eavesdropping adversary still performs significantly better than random guessing. This indicates that even the defended Bluetooth traffic traces contain useful information for an adversary. At the same time, the costs introduced by the defenses are high: to achieve their small levels of protection they introduce additional traffic and/or delays reaching an overhead in the range of 1× to 23× and delays up to 1 sec per packet. This raises a question about the applicability of such defenses for Bluetooth applications running on current wearable devices. Furthermore, we find that the various defenses behave differently, depending on the adversarial task. In particular, our results show that defending against application or action identification is somewhat easier

than device identification, thus indicating that global traffic patterns are the hardest to hide. Additionally, our evaluation shows that the defenses are not *fair*: we find that they do not provide the same level of protection across applications or actions (for instance, apps that communicate a lot are not well protected) and their costs are variable across applications or actions (we observe that applications that stream information have high padding costs, *e.g.,* `Camera` and fitness applications for workout monitoring). Finally, our empirical evaluation of these defenses confirms the robustness of our attacking methodology as depending on the adversarial task and the defense, our classifier adapts its important features. For instance, Bluetooth Low Energy device identification relied mostly on packet sizes (Figure 2.5b), unlike Bluetooth Classic that used mostly timings (Figure 2.5a). However, when we apply per-packet padding to the Low Energy traces, the classifier adjusts and gives higher importance to timings (Appendix A.4, Figure A.3a). Overall, our experimental results highlight the need for the design and evaluation of novel approaches for defending against traffic-analysis attacks on Bluetooth communications.

2.8 Discussion & Limitations

2.8.1 Discussion

Impact of the Attacks. The traffic-analysis attacks presented in this work can be used to infer information from Bluetooth communications, despite the use of encryption.

Device identification enables tracking users only by observing encrypted communications, thus defeating MAC address randomization. This does not require observing any pairings / paging events or plaintext identifiers. Device identification can also facilitate active attacks by revealing the model and version of a communicating device. Advertisers already use Bluetooth and Wi-Fi signals to passively and actively locate users (*e.g.,* consumers in a store) [156, 242].

Similarly, application and action fingerprinting leak sensitive actions performed by the wearer, *e.g.,* the recording of an insulin injection or a heartbeat measurement. On a side note, Apple Watch has an "arrhythmia alert" feature that continuously measures the heart beat on the watch and sends notifications to the phone in case of irregular patterns that could indicate a stroke. We could not simulate arrhythmia events, but all the evidence we have from our experiments suggests that such an action could be fingerprintable without an appropriate defense mechanism. Therefore, a passive observer could identify users' susceptibility to heart attacks over the Bluetooth network, despite the encryption. Finally, application-opening identification and action identification can be exploited to build user profiles and to serve targeted advertisements, as it is already the case with Bluetooth-based "proximity advertising" [7, 135, 196].

Cost of the Attack. The overall cost of the attack consists of purchasing a set of devices of interest (including both wearable devices and smartphones). These devices are consumer-grade hardware that have accessible prices. We also showed in §2.6.2.2 that the adversary does not have to train on every combination of devices and applications. We suspect that after collecting data from enough devices, actions on new hardware can become classified without the overhead of training. A counter-argument is the *aging* of the dataset, which could force the adversary to re-train often. We briefly demonstrate in Appendix, Section A.2 how a dataset can be used over at least a month, but further study is needed to understand how quickly the usefulness of a dataset degrades. In other domains such as website fingerprinting, attacks have

been successful with datasets that were several years old [200]. We expect wearable devices' firmware, OSes and applications to change at a slower rate than websites.

Bluetooth Sniffing Technologies. The adversary also needs a reliable Bluetooth sniffer. The most accurate models are so-called "wide-band" scanners (*e.g.,* the Ellisys Vanguard [83] or the Frontline Sodera [202]). These models ignore the Bluetooth frequency hopping and concurrently capture the traffic of all channels. The complexity and broad functionality of these devices comes at a high price (≈ 50K USD). However, recent research has demonstrated that similar results can be achieved using less expensive Software-Defined Radios (SDRs) [57, 58, 206]. For instance, Cominelli *et al.* built an SDR sniffer that works on a single Ettus N310 board (≈ 10K USD) or two Ettus B210 boards (2 × 2,000 USD) [57]. Finally, there also exists a consumer-grade class of cheaper, less accurate Bluetooth sniffers (*e.g.,* Ubertooth, ≈ 100 USD). They only listen on one channel at a time. These low-end scanners attempt to *follow* an active connection by brute-forcing the hopping pattern parameters [185]. When successful, this enables an inexpensive scanner to accurately capture all traffic simply by "hopping along" with the pair of communicating devices. In practice today, this process is still imprecise and many packets are missed. Nonetheless, researchers have shown that using two synchronized Ubertooth scanners leads to improved Bluetooth traffic capture rate [4, 5]. Although this work uses a commercial Bluetooth sniffer, there is an ongoing research trend focusing on less expensive, accurate Bluetooth sniffing.

Impact of Packet Loss on the Attacks. In our experiments, we use a high-end sniffer that is co-located with the target devices. In this configuration, the sniffer has close to 100% packet capture rate. In practice, lower-end sniffers will suffer packet loss. Collisions from other devices and the distance between the target and the eavesdropper also increase loss. In Appendix A.3, we briefly investigate how the attack accuracy varies with degraded capture conditions. We observe that even with 50% packet loss, the loss in accuracy is only 10 percentage points (Appendix A.3, Figure A.1), with the mean accuracy dropping from 64% to 54%. For high-volume apps, the mean accuracy drops from 90% to 77%. This experiment indicates that the approach is robust to packet losses: even when missing every other packet, the attacker is able to classify with significant accuracy. This observation suggests that the attack could also be performed with inexpensive Bluetooth sniffers.

Other Attacks. There exist a number of active attacks that can break the confidentiality of Bluetooth communications [12, 14, 223, 239, 245]. There are also attacks against the MAC randomization mechanism employed by the Bluetooth protocol [28, 28, 44, 154, 247]. Currently, these attacks are more economical to run than our traffic-analysis attacks that require a significant effort for training the adversarial machine-learning classifiers. However, these attacks have already received significant visibility, and we expect that the Bluetooth Special Interest Group and device manufacturers will soon take them into account and apply the necessary countermeasures. We remark that our approach is complementary to these attacks and will be applicable even when the above attacks are patched. Finally, we remind the reader that unlike these works, ours considers a weaker adversary who passively observes ongoing communications.

Implementation of Defenses. A defense against traffic analysis could be implemented at different layers of the stack involved with Bluetooth communications: the Bluetooth protocol, the OS, or the applications. An implementation in a lower layer, *e.g.,* the Bluetooth stack, would provide application transparency, but specifying a single defense strategy that works across devices and applications without a prohibitive cost seems challenging. On the con-

trary, application developers could protect against in-application actions fingerprinting by enumerating the data exchanges and ensuring that the traffic from sensitive actions "blends-in". In this case, all sensitive actions would need to have the same patterns as some other non-sensitive actions (or ideally, all other actions). Nonetheless, such an approach would only provide local (*i.e.,* in-application) protection. To make the fingerprinting of applications and cross-application actions more difficult, the various developers would need to coordinate with each other. Otherwise, a defense can easily become itself a fingerprint if only one or a few applications implement it. Therefore, another interesting possibility is to create "anti traffic-analysis policies" in the OS of wearable devices. Apps could request a particular defense strategy that matches their requirements in terms of latency, bandwidth, and battery usage. Meanwhile, the operating system could standardize and maintain defense strategies transparently, making their deployment easy for developers.

2.8.2 Limitations

Moderate Testbed Size. Our testbed consists of only 13 devices. Although this number is modest, our experiments incurred significant human costs in operating these non-automatable devices. Our primary dataset consists of 96 hours of raw recording. Each of the 2,361 30-second samples was recorded by a human and required sometimes minutes of resetting the devices, not to mention performing the physical activities corresponding to the action captured. We made a best-effort to cover a comprehensive and diverse set of wearable devices from popular vendors. We are hopeful that future work will further generalize the attack to more devices, which we could only hint towards with our transferability experiment (§2.6.2.2).

Closed-world Scenario. Our setting corresponds to a "closed-world" scenario where the adversary can model all possible actions/applications of the devices available in our testbed. This could be justified due to the small number of applications currently available on wearable devices. We leave as an interesting future work the task to explore the performance of the attacks "in the wild", *e.g.,* by collecting real Bluetooth traffic traces around a campus or a gym and attempting to classify them.

Method Scalability. We demonstrated an example in which a model trained on a pair of devices can be used for classifying actions of other devices (§2.6.2.2). However, this experiment has been limited to two pairs of devices, and it is unclear whether a unique model could be successfully trained to classify actions originating from many classes of devices. While we expect it to be the case for similar devices, our preliminary results show that a model trained to recognize applications on an Apple watch does not perform well when used to recognize applications on an Android device, highlighting that an adversary should take into account heterogeneous devices when training her classifier.

Environment of the Capture. In this work, we did not experiment with the range of capture and kept the sniffer close-by to the devices (max ≈2 m). Bluetooth Classic and Low Energy have a maximum theoretical range that greatly varies depending on the Bluetooth flavor, the encoding, the sender/receiver's antenna gains and the transmitted power [33]. For consumer devices, the range under optimal conditions is between 50 and 100 m and, we estimate, from meters to tens of meters under realistic conditions. Furthermore, our attacks were conducted in a single environment that is fairly noisy (with tens of active Wi-Fi and Bluetooth devices in vicinity). We suspect that a less noisy environment (*e.g.,* a home, in the case of a nosy neighbor) would produce cleaner traces that are easier to train upon, facilitating the attacks, but further study is needed to understand the impact of noise and collisions on traffic analysis.

2.9 Related Work

Bluetooth Eavesdropping. The first open-source Bluetooth Sniffer was BlueSniff [203]: This work demonstrates how to retrieve the MAC address of communicating Bluetooth Classic devices and how to recover the hopping sequence. Similar results are later shown on Bluetooth Low Energy [185]: using an Ubertooth device, Mike Ryan showed how to recover the hopping sequence and eavesdrop on a single BLE connection. The author also demonstrates that a pairing done with JustWorks or a 6-digit PIN can be decrypted. Subsequently, Albazrqaoe *et al.* used two synchronized Ubertooth devices to obtain a capture accuracy greater than a single Ubertooth [4, 5]. Finally, Cominelli *et al.* rely on software-defined radio (SDR) to concurrently capture Bluetooth Classic traffic on all channels [57], at a lower cost than full-band commercial sniffers. Their most recent work uses a GPU to process BLE traffic in real-time [58].

Bluetooth Traffic Analysis. There exist few related works that perform traffic-analysis attacks on Bluetooth communications. This is possibly due to the non-existence of reliable, inexpensive Bluetooth sniffing tools in the past. Closest to ours is the work by Das *et al.* [69]. They focus on 6 Bluetooth Low Energy fitness trackers in a gym. First, they demonstrate that BLE traffic is correlated with the wearer's movements, thus making it possible to infer if the wearer is idle, walking, or running. Second, they show how the traffic is linked to the gait of the wearer, and that the encrypted traffic is enough to recognize a person with 97.6% accuracy across 10 users. Acar *et al.* infer user actions in a smart home using a layered traffic-analysis attack [2]. Their methodology is similar to ours: they first perform device identification and then use it as a stepping stone to further infer device states and user activities. However, their IoT testbed consists of only one device that communicates using Bluetooth (a smart BLE light bulb).

Bluetooth Device Fingerprinting/Tracking. Several works focus on Bluetooth device fingerprinting, *i.e.,* device identification and tracking. Their goal is either to propose an authentication mechanism, *e.g.,* to identify MAC spoofing, or to demonstrate an attack, *e.g.,* BLE device tracking despite the MAC address rotation. Our device-identification attack (§2.5) falls into the second category; however, we only use it as a first step towards the rest of our contributions (application and user-action identification) that are orthogonal to this category of related works.

On the defense side, Aksu *et al.* create a testbed composed of 6 Bluetooth Classic smartwatches connected to a single smartphone, and they demonstrate that the smartwatches can be identified via their communications' timings [3]. However, their model is different and they do not sniff Bluetooth traces in the air, rather collect them using the Bluetooth Host Controller Interface (HCI) log on the smartphone. Huang *et al.* propose BlueID [120], a system that prevents identity spoofing by fingerprinting the clock of the master device.

Concerning the attacks, Zuo *et al.* demonstrate that Bluetooth Low Energy devices can be recognized by plaintext identifiers found in their communications [247]. They suggest application- or protocol-level solutions to better rotate static identifiers. Their solutions would not thwart our device identification attack that works on encrypted traffic (§2.5). Becker *et al.* use static identifiers differently [28]: they demonstrate that the rotation of MAC address and other static identifiers are not synchronized, which enables defeating the MAC randomization. More generally, Celosia and Cunche examine BLE devices in the wild and show that they fail to properly rotate their MAC addresses, thus enabling tracking [44]. Then, Korolova and Sharma show that the "nearby devices" list that Bluetooth devices maintain and which most applications can obtain, can be used to track users across applications [132]. Targeting more specifically Apple's Continuity protocol, Martin *et al.* reverse-engineer the protocol and find flaws that

defeat MAC address randomization and that leak information about the device types and user activities [154]. Similarly, Celosia and Cunche also reverse-engineer the protocol and demonstrate how it reveals information about human activities in a smart home [45].

Other Bluetooth Attacks and Tools. There exist other attacks on the Bluetooth protocol that are orthogonal to our work; for instance, active attacks, or protocol-specific attacks on wearable devices. Fixing these will likely not affect our higher-level attacks that rely on Bluetooth communication metadata.

We first list attacks on the Bluetooth protocol and implementations. Antonioli *et al.* demonstrate how to break the key negotiation protocol of Bluetooth Classic [14], forcing it to a 1-byte entropy encryption key. The same authors reverse-engineer and identify vulnerabilities in Google Nearby Connections [11], a protocol that uses a combination of Bluetooth Classic, Low Energy, and Wi-Fi for short-distance transfers. Then, they also exploit role-switching (slave/-master) and legacy pairings to perform a Man-in-the-Middle on Bluetooth Classic [12]. Wu *et al.* present an active spoofing attack on Bluetooth Low Energy by exploiting the reconnection procedure [239], whereas Wang *et al.* demonstrate an active attack to bypass Bluetooth Low Energy authentication and encryption [223]. Finally, Zhang *et al.* show a downgrade attack based on Bluetooth Low Energy's Secure Connection Only (SCO) mode [245].

In the category of Bluetooth tools, Mantz *et al.* present InternalBlue [153], a Bluetooth experimentation framework that enables patching the Bluetooth firmware of Broadcom chips. Similarly, Classen and Hollick show how to analyze Bluetooth communications using consumer devices [54], while Ruge *et al.* design an emulation framework and perform fuzzing to uncover vulnerabilities [184]. In particular, the authors find unattended Remote Code Executions (RCE) on some Bluetooth chips.

Focusing on wearable devices, Classen *et al.* perform an in-depth security analysis of the Fitbit ecosystem [55], analyzing the firmware and the application of a fitness tracker. They find vulnerabilities that enable flashing malware, disabling encryption, and extracting private information about the users. Hilts *et al.* perform a comparative analysis of the security and privacy of fitness trackers [116]; they highlight security vulnerabilities and issues with their data policies.

2.10 Conclusion

In this work, we have shown that encrypted Bluetooth communications between a wearable device and its connected smartphone leak information about their contents: a passive adversary can infer sensitive information by exploiting their metadata via traffic-analysis attacks. Our empirical evaluation on a Bluetooth (Classic and Low Energy) traffic dataset generated by a diverse set of wearable devices demonstrates that an eavesdropper can accurately identify communicating devices to their model number, recognize user activities (*e.g.*, health monitoring or exercising), the opening of specific applications on smartwatches, and fine-grained user actions (*e.g.*, recording an insulin injection), and extract the profile and habits of the wearer. Our experimental analysis of common defense strategies against our traffic-analysis attacks, such as padding or delaying packets, and injecting dummy traffic, show that these do not provide sufficient protection, and that they introduce significant costs for Bluetooth communications. Overall, this research highlights an open problem regarding the confidentiality of Bluetooth communications and the need for designing novel efficient defenses to address it.

3. Reducing Metadata Leakage from Static Objects

In this chapter, we propose a new ciphertext format designed to minimize metadata leakage. The traditional ciphertext formats (*e.g.,* PGP, TLS) typically contain plaintext metadata that help a recipient decrypt in multiple scenarios: with symmetric or asymmetric keys, with multiple recipients, with multiple cryptographic suites. However, these metadata also reveal information to third parties. We demonstrate that this leakage to third parties is not needed: the same functionality can be achieved with the concept of PURBs, Padded Uniform Random Blobs that do not leak any metadata except their length. PURBs employ Padmé, a novel padding scheme that limits information leakage via ciphertexts of maximum length M to a practical optimum of $O(\log\log M)$ bits, comparable to padding to a power of two, but with lower overhead of at most 12% and decreasing with larger payloads.

Scope. In this chapter, we focus exclusively on the leakage of a single file or network message that we regroup under the notion of an encrypted *blob* or encrypted byte array. When used in a larger system (*e.g.,* in a communication system with many participants), transmitting a blob has additional metadata, which we explore in Chapter 4.

Acknowledgments. This work is a collaboration between Kirill Nikitin and Ludovic Barman: both authors contributed equally [161]. To ensure non-overlapping contents in the respective doctoral theses, the results are divided between the two authors. The technical contribution in PURBs is composed of two distinct components: an encoding scheme and a padding scheme. In this thesis, we do not describe the former, rather we focus on the latter. To properly convey the contribution, the introduction, motivations and related work present the whole project.

3.1 Introduction

Traditional encryption schemes and protocols aim to protect only their data payload, leaving related metadata exposed. Formats such as PGP [246] reveal in cleartext headers the public keys of the intended recipients, the algorithm used for encryption, and the actual length of the payload. Secure-communication protocols similarly leak information during key and algorithm agreement. The TLS handshake [178], for example, leaks in cleartext the protocol version, chosen cipher suite, and the public keys of the parties. This metadata exposure is traditionally assumed not to be security-sensitive, and it facilitates the decryption for the intended recipient(s).

Research has consistently shown, however, that attackers can exploit metadata to infer sensitive information about communication content. In particular, an attacker may be able to fingerprint users [165, 218] and the applications they use [244]. Using traffic analysis [66],

an attacker may be able to infer websites a user visited [66, 82, 164, 226, 227] or videos a user watched [174, 175, 189]. On VoIP, metadata can be used to infer the geolocation [147], the spoken language [236], or the voice activity of users [46]. Side-channel leaks from data compression [128] facilitate several attacks on SSL [29, 98, 182]. The lack of proper padding might enable an active attacker to learn the length of the user's password from TLS [222] or QUIC [180] traffic. In social networks, metadata can be used to draw conclusions about users' actions [108], whereas telephone metadata has been shown to be sufficient for user re-identification and for determining home locations [155]. Furthermore, by observing the format of packets, oppressive regimes can infer which technology is used and use this information for the purposes of incrimination or censorship. Most TCP packets that Tor sends, for example, are 586 bytes due to its standard cell size [115].

As a step towards countering these privacy threats, we propose that encrypted data formats should produce *Padded Uniform Random Blobs* or PURBs: ciphertexts designed to protect all encryption metadata. A PURB encrypts application content and metadata into a single blob that is indistinguishable from a random string, and is padded to minimize information leakage via its length while minimizing space overhead. Unlike traditional formats, a PURB does not leak the encryption schemes used, who or how many recipients can decrypt it, or what application or software version created it. While simple in concept, because PURBs by definition contain no cleartext structure or markers, encoding and decoding them efficiently presents practical challenges.

This first key contribution is an encoding scheme that supports any number of recipients, who can use either shared passwords or public-private key pairs utilizing multiple cryptographic suites. The main technical challenge is providing efficient decryption to recipients without leaving any cleartext markers. If efficiency was of no concern, the sender could simply discard all metadata and expect the recipient to parse and trial-decrypt the payload using every possible format version, structure, and cipher suite imaginable. However, real-world adoption requires both decryption efficiency and cryptographic agility. The encoding scheme combines a variable-length header containing encrypted metadata with a symmetrically-encrypted payload. The header's structure enables efficient decoding by legitimate recipients via a small number of trial decryptions. It facilitates the seamless addition and removal of supported cipher suites, while leaking no information to third parties without a decryption key.

To reduce information leakage from data lengths, the second main contribution is Padmé, a padding scheme that groups encrypted PURBs into indistinguishability sets whose visible lengths are representable as limited-precision floating-point numbers. Like obvious alternatives such as padding to the next power of two, Padmé reduces maximum information leakage to $O(\log\log M)$ bits, where M is the maximum length of encrypted blob a user or application produces. Padmé greatly reduces constant-factor overhead with respect to obvious alternatives, however, enlarging files by at most +12%, and less as file size increases.

Our evaluation shows that the performance of decoding a PURB is similar to that of PGP; at the same time, unlike PGP, the encrypted PURB does not reveal which cryptographic parameters are used. Creating a PURB is slower than creating a PGP ciphertext, due to the header construction being more involved.

Regarding Padmé, we evaluate its performance with real-world datasets. Our analysis shows that many objects are trivially identifiable by their unique sizes without padding, or even after padding to a fixed block size (*e.g.*, that of a block cipher or a Tor cell). We show that Padmé can significantly reduce the number of objects uniquely identifiable by their sizes: from 83% to 3%

for 56k Ubuntu packages, from 87% to 3% for 191k YouTube videos, from 45% to 8% for 848k hard-drive user files, and from 68% to 6% for 2.8k websites from the Alexa top 1M list. This stronger leakage protection incurs an average space overhead of only 3%.

In summary, the contributions of this section are as follows:

- We introduce PURBs, a ciphertext format that reveals no metadata information to observers without decryption keys, while efficiently supporting multiple recipients and cipher suites.
- We introduce Padmé, a padding scheme that asymptotically minimizes information leakage from data lengths while also limiting size overheads.

3.2 Motivation and Applications

Our goal is to define a generic method applicable to most of the common data-encryption scenarios such that the techniques are flexible to the application type, to the cryptographic algorithms used, and to the number of participants involved. We also seek to enhance plausible deniability such that a user can deny that a PURB is created by a given application or that the user owns the key to decrypt it. We envision several immediate applications that could benefit from using PURBs.

E-mail Protection. E-mail systems traditionally use PGP or S/MIME for encryption. Their packet formats [43], however, exposes format version, encryption methods, number and public-key identities of the recipients, and public-key algorithms used. In addition, the payload is padded only to the block size of a symmetric-key algorithm used, which does not provide "size privacy", as we show in §3.5. Using PURBs for encrypted e-mail could minimize this metadata leakage. Furthermore, as e-mail traffic is normally sparse, the moderate overhead PURBs incur can easily be accommodated.

Initiation of Cryptographic Protocols. In most cryptographic protocols, initial cipher suite negotiation, handshaking, and key exchange are normally performed unencrypted. In TLS 1.2 [76], an eavesdropper who monitors a connection from the start can learn many details such as cryptographic schemes used. The eavesdropper can also fingerprint the client [181] or distinguish censorship-circumvention tools that try to mimic TLS traffic [92, 118]. TLS 1.3 [178] takes some protective measures by reducing the amount of unencrypted metadata during the handshake.These measures are only partial, however, and leave other metadata, such as protocol version number, cipher suites, and public keys, still visible. PURBs could facilitate fully-encrypted handshaking from the start.

3.3 PURB Construction

As stated in the introduction of the chapter, this section is only a brief overview of the construction technique of a PURB. We refer the reader to the paper for the full contribution [161].

Preliminaries. Let λ be a standard security parameter. Let S be cipher suite defined as

$$S = \langle \mathbb{G}, p, g, \texttt{Hide}(\cdot), \Pi, \mathrm{H}, \hat{\mathrm{H}} \rangle$$

where:

- $\mathbb{G}$ is a cyclic group of order p generated by g where the gap-CDH problem is hard to solve (*e.g.*, an elliptic curve or a multiplicative group of integers modulo a large prime);
- `Hide` is a mapping: $\mathbb{G} \to \{0,1\}^\lambda$ that encodes a group element of $\mathbb{G}$ to a binary string that is indistinguishable from a uniform random bit string (*e.g.*, Elligator [31], Elligator Squared [18, 211]);
- $\Pi = (\mathcal{E}, \mathcal{D})$ is an ind$-cca2-secure authenticated-encryption scheme [30] where $\mathcal{E}_K(m)$ and $\mathcal{D}_K(c)$ are encryption and decryption algorithms, respectively, given a message m, a ciphertext c, and a key K.
- $\mathrm{H} : \mathbb{G} \to \{0,1\}^{2\lambda}$ and $\hat{\mathrm{H}} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ are two distinct cryptographic hash functions.

PURB Definition. Then, we define the four algorithms PurbEnc = (Setup, KeyGen, Enc, Dec) as follows:

- $\mathsf{Setup}(1^\lambda) \to S$: Create a cipher suite $S = \langle \mathbb{G}, p, g, \mathtt{Hide}(\cdot), \Pi, \mathrm{H}, \hat{\mathrm{H}} \rangle$.
- $\mathsf{KeyGen}(S) \to (sk, pk)$: Create a key pair $(sk(S), pk(S))$.
- $\mathsf{Enc}(R, m) \to c$: Given a set of public keys $R = \{pk_1(S_1), \ldots, pk_r(S_r)\}$, each on an indicated suite, and a message m, output a ciphertext c.
- $\mathsf{Dec}(sk(S), c) \to m/\perp$: Given a valid ciphertext c and a valid decryption key $sk(S)$, return m. Return $\perp$ otherwise.

In the related paper [161], we present MsPURB, a instantiation of a PurbEnc algorithm. In the case of one recipient using a public key, the MsPURB is equivalent to Integrated Encryption Scheme (IES) [1]; it also supports multiple recipients each using their own suite. We show that MsPURB is ind$-cca2-secure against an adversary that does not possess a decryption key.

The key aspect of a PURB is that c can be efficiently decrypted by any valid sk in any suite S, without additional metadata. To achieve this, the PGP header format uses plaintext information to indicate the position of group elements and nonces. On the contrary, the ciphertext c is not accompanied by any header, and hence reveals no information to the adversary (except its length $|c|$).

3.4 Padmé Construction

The encoding scheme PurbEnc (§3.3) produces blobs of data that are indistinguishable from random bit-strings of the same length, thus leaking no information to the adversary directly via their content. The length itself, however, might indirectly reveal information about the content. Such leakage is already used extensively in traffic-analysis attacks, *e.g.*, website fingerprinting [82, 164, 226, 227], video identification [174, 175, 189], and VoIP traffic fingerprinting [46, 236]. Although solutions involving application- or network-level padding are numerous, they are typically designed for a specific problem domain, and the more basic problem of length-leaking ciphertexts remains. In any practical solution, some leakage is unavoidable. We show, however, that typical approaches such as padding to the size of a block cipher are fundamentally insufficient for efficiently hiding the plaintext length effectively, especially for plaintexts that may vary in size by orders of magnitude.

We introduce Padmé, a novel padding scheme designed for, though not restricted to, encoding PURBs. Padmé reduces length leakage for a wide range of encrypted data types, ensuring asymptotically lower leakage of $O(\log\log M)$, rather than $O(\log M)$ for common stream- and block-cipher-encrypted data. Padmé's space overhead is moderate, always less than 12% and decreasing with file size. The intuition behind Padmé is to pad objects to lengths representable as limited-precision floating-point numbers. A Padmé length is constrained in particular to

have no more significant bits (*i.e.,* information) in its mantissa than in its exponent. This constraint limits information leakage to at most double that of conservatively padding to the next power of two, while reducing overhead through logarithmically-increasing precision for larger objects.

Scope. Many defenses already exist for specific scenarios, *e.g.,* against website fingerprinting [82, 228]. Padmé does not attempt to compete with tailored solutions in their domains. Instead, Padmé aims for a substantial increase in application-independent length-leakage protection as a generic measure of security and privacy.

3.4.1 Definitions

We design Padmé again using intermediate strawman approaches for clarity. To compare these straightforward alternatives with our proposal, we define a game where an adversary guesses the plaintext behind a padded encrypted blob. This game is inspired by related work such as defending against a "perfect attacker" [228].

Padding Game. Let P denote a finite collection of plaintext objects of maximum length M: *e.g.,* data, documents, or application data-units. Let $f : \mathbf{N} \to \mathbf{N}$ be a padding function that yields the padded size $|c|$ given a plaintext length $|p|$, for $p \in P$.

An honest user chooses a plaintext $p \in P$, then pads and encodes it into a PURB c such that $|c| = f(|p|)$. The adversary knows almost everything: all possible plaintexts P, the PURB c and the public parameters used to generate it, such as schemes and the recipients. The adversary lacks only the private inputs and decryption keys for c. The adversary's goal is to guess the plaintext p based on the observed PURB c of length $|c|$.

Overhead. We define the additive overhead of $|c|$ over $|p|$to be $|c| - |p|$, the number of extra bytes added by padding. The multiplicative overhead of padding is $\frac{|c|-|p|}{|p|}$, the relative fraction by which $|c|$ expands $|p|$.

Leakage. Let R be the image set of f; R consists of the padded lengths that f can produce from plaintexts in P. We quantify the leakage of padding function f in terms of the number of elements in R. More precisely, we define the leakage as the number of bits required to specify a unique element of R, which is $\lceil \log_2 |R| \rceil$.

Intuitively, a function that pads everything to a constant size larger than all plaintexts (*e.g.,* $f(p) = \max_p |p|$) leaks no information to the adversary, because $|R| = 1$ and observing $|c|$ leaks no information about the plaintext: every one of them could have produced $|c|$. On the contrary, more fine-grained padding functions leak more bits.

Design Goals. Our goal in designing the padding function is to manage both space overhead from padding and maximum information leaked to the adversary.

3.4.2 Strawman Padding Approaches

We first explore two strawman designs, based on different padding functions f. A padding function that offers any useful protection cannot be one-to-one, otherwise the adversary could trivially invert it and recover $|p|$. We also exclude randomized padding schemes for simplicity, and because adversaries may be able to cancel out and defeat random padding statistically

over many observations. Therefore, only padding functions that group many plaintext lengths into fewer padded ciphertexts are of interest in our analysis.

Strawman 1: Fixed-Size Blocks. We first consider a padding function $f(L) = b \cdot \lceil L/b \rceil$, where b is a block size in bytes. This is how objects often get "padded" by default in practice, *e.g.,* in block ciphers or Tor cells. In this case, the ciphertext size is a multiple of b, the maximum additive overhead incurred is $b-1$ bytes, and the leakage is $\lceil \log_2 M/b \rceil = O(\log M)$, where M is the maximum plaintext size.

In practice, when plaintext sizes differ by orders of magnitude, there is no good value for b that serves all plaintexts well. For instance, consider b =1 MB. Padding small files and network messages would incur a large overhead: *e.g.,* padding Tor's 512 B cells to 1 MB would incur overheads of 2000×. In contrast, padding a 700 MB movie with at most 1 MB of chaff would add only a little confusion to the adversary, as this movie may still be distinguishable from others by length.

Therefore, to reduce information leakage asymptotically over a vast range of cleartext sizes, the padding must depend on plaintext size.

Strawman 2: Padding to Powers of 2. The next step is to pad to varying-size blocks, which is the basis for our actual scheme. The intuition is that for small plaintexts, the blocks are small too, yielding modest overhead, whereas for larger files, blocks are larger and group more plaintext lengths together, improving leakage asymptotically. A simple approach is to pad plaintexts into buckets b_i of size varying as a power of some base, *e.g.,* two, so $b_i = 2^i$. The padding function is thus $f(L) = 2^{\lceil \log_2 L \rceil}$. We call this strawman NextP2.

Because NextP2 pads plaintexts of maximum length M into at most $\lceil \log_2 M \rceil$ buckets, the image R of f contains only $O(\log M)$ elements. This represents only $O(\log \log M)$ bits of entropy or information leakage, a major asymptotic improvement over fixed-size blocks. However, the maximum overhead is substantial: almost +100%. This implies that a 17 GB Blu-ray movie would be padded into 32 GB.

Using powers of another base $x > 2$, we reduce leakage further at a cost of more overhead: *e.g.,* padding to the nearest power of 3 incurs overhead up to +200%, with less leakage but still $O(\log \log M)$. We could reduce overhead by using a fractional base $1 < x < 2$, but fractional exponents are cumbersome in practical padding functions. Although this second strawman succeeds in achieving asymptotically lower leakage than padding to fixed-size blocks, it is less attractive in practice.

3.4.3 Padmé

We now describe our padding scheme Padmé, which limits information leakage about the length of the plaintext for wide range of encrypted data sizes. Similarly to the previous strawman, Padmé also asymptotically leaks $O(\log \log M)$ bits of information, but its overhead is much lower (at most 12% and decreasing with L).

Intuition. In NextP2, any permissible padded length L has the form $L = 2^n$. We can therefore represent L as a binary floating-point number with a $\lfloor \log_2 n \rfloor + 1$-bit exponent and a mantissa of zero, *i.e.,* no fractional bits.

In Padmé, we similarly represent a permissible padded length as a binary floating-point number, but we allow a non-zero mantissa at most as long as the exponent (see Figure 3.2).

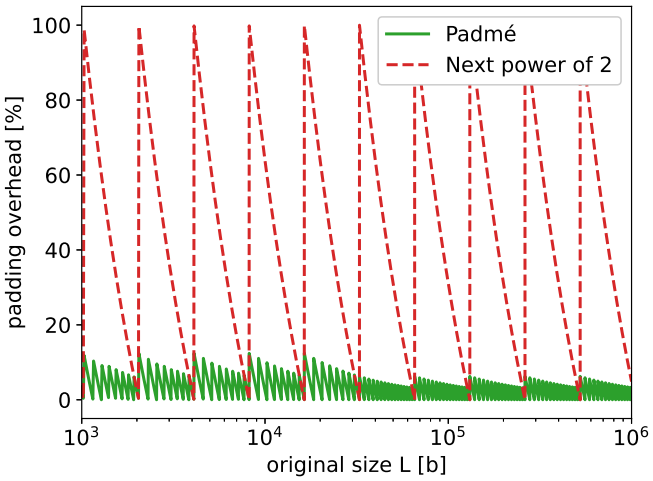
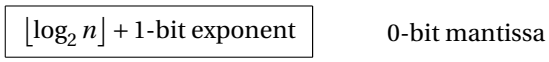


Figure 3.1 – Maximum multiplicative expansion overhead with respect to the plaintext size L. The naïve approach to pad to the next power of two has a constant maximum overhead of 100%, whereas Padmé's maximum overhead decreases with L, following $\frac{1}{2\log_2 L}$.



In the strawman NextP2, the allowed length $L = 2^n$ can be represented as a binary floating-point number with a $\lfloor \log(n) + 1 \rfloor$ bits of exponent and no mantissa.

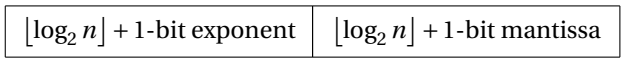


Figure 3.2 – Padmé represents lengths as floating-point numbers, allowing the mantissa to be of at most $\lfloor \log_2 n \rfloor + 1$ bits.

This approach doubles the number of bits used to represent an allowed padded length — hence doubling absolute leakage via length — but allows for more fine-grained buckets, reducing overhead. Padmé asymptotically leaks the same number of bits as NextP2, differing only by a constant factor of 2, but reduces space overhead by almost 10× (from +100% to +12%). More importantly, the multiplicative expansion overhead decreases with L (see Figure 3.1).

Algorithm. We provide the pseudocode for Padmé in Algorithm 3.1. To compute the padded size $L' = f(L)$, ensuring that its floating-point representation fits in at most $2 \times \lfloor \log_2 n \rfloor + 1$ bits, we require the last $E - S$ bits of L' to be 0. $E = \lfloor \log_2 L \rfloor$ is the value of the exponent, and $S = \lfloor \log_2 E \rfloor + 1$ is the size of the exponent's binary representation. The reason for the substraction will become clear later. For now, we demonstrate how E and S are computed in Table 3.1.

Recall that Padmé requires the mantissa's bit length to be no longer than that of the exponent. In Table 3.1, for the value $L = 9$ the mantissa is longer than the exponent: it is "too precise" and therefore not a permitted padded length. The value 10 is permitted, however, so a 9 byte-long ciphertext is padded to 10 bytes.

To understand why Padmé requires the low $E - S$ bits to be 0, notice that forcing all the last E bits to 0 is equivalent to padding to a power of two. In comparison, Padmé allows S extra bits to represent the padded size, with S defined as the bit length of the exponent.

Algorithm 3.1: Padmé

Data: length of content L
Result: length of padded content L'

$E \leftarrow \lfloor \log_2 L \rfloor$ // L's floating-point exponent
$S \leftarrow \lfloor \log_2 E \rfloor + 1$ // # of bits to represent E
$z \leftarrow E - S$ // # of low bits to set to 0
$m \leftarrow (1 \ll z) - 1$ // mask of z 1's in LSB
$L' \leftarrow (L + m)$ & $\sim m$ // round up using mask m to clear last z bits

Table 3.1 – The IEEE floating-point representations of 8, 9 and 10. The value 8 has 1 bit of mantissa (the initial 1 is omitted), and 2 bits of exponents; 9 has a 3-bits mantissa and a 2-bit exponent, while the value 10 as 2 bits of mantissa and exponents. Padmé enforces the mantissa to be no longer than the exponent, hence 9 gets rounded up to the next permitted length 10.

L	L	E	S	IEEE representation	
8	0b1000	3	2	0b1.0	* 2^0b11
9	0b1001	3	2	0b1.001	* 2^0b11
10	0b1010	3	2	0b1.01	* 2^0b11

Leakage and Overhead. By design,if the maximum plaintext size is M, Padmé's leakage is $O(\log\log M)$ bits, the length of the binary representation of the largest plaintext. As we fix $E-S$ bits to 0 and round up, the maximum overhead is $2^{E-S}-1$. We can estimate the maximum multiplicative overhead as follows:

$$\text{max overhead} = \frac{2^{E-S}-1}{L} < \frac{2^{E-S}}{L} \approx \frac{2^{\lfloor \log_2 L \rfloor - \lfloor \log_2 \log_2 L \rfloor - 1}}{L} \approx \frac{1}{2 \cdot 2^{\log_2 \log_2 L}} = \frac{1}{2\log_2 L} \quad (3.1)$$

Thus, Padmé's maximum multiplicative overhead decreases with respect to the file size L. The maximum overhead is $+11.\overline{11}\%$, when padding a 9-byte file into 10 bytes. For bigger files, the overhead is smaller.

On Optimality. There is no clear sweet spot on the leakage-to-overhead curve. We could easily force the last $\frac{1}{2}(E-S)$ bits to be 0 instead of the last $E-S$ bits, for example, to reduce overhead and increase leakage. Still, what matters in practice is the relationship between L and the overhead. We show in §3.5 how this choice performs with various real-world datasets.

3.5 Evaluation

In evaluating a padding scheme, one important metric is overhead incurred in terms of bits added to the plaintexts. By design, Padmé's overhead is bounded by $\frac{1}{2\cdot\log_2 L}$. As discussed in §3.4.3, Padmé does not escape the typical overhead-to-leakage trade-off, hence Padmé's novelty does not lie in this trade-off. Rather, the novelty lies in the practical relation between L and the overhead. Padmé's overhead is moderate, at most +12% and much less for large PURBs.

A more interesting question is how effectively, given an arbitrary collection of plaintexts P, Padmé hides which plaintext is padded. Padmé was designed to work with an arbitrary collection of plaintexts P. It remains to be seen how Padmé performs when applied to a specific set of plaintexts P, *i.e.,* with a distribution coming from the real world, and to establish

Table 3.2 – Datasets used in the evaluation of anonymity provided by Padmé.

Dataset	# of objects
Ubuntu packages	56,517
YouTube videos	191,250
File collections	3,027,460
Alexa top 1M websites	2,627

how well it groups files into sets of identical length. In the next section, we experiment with four datasets made of various objects: a collection of Ubuntu packages, a set of YouTube videos, a set of user files, and a set of Alexa Top 1M websites.

3.5.1 Datasets and Methodology

The Ubuntu dataset contains 56,517 unique packages, parsed from the official repository of a live Ubuntu 16.04 instance. As packages can be referenced in multiple repositories, we filtered the list by name and architecture. The reason for padding Ubuntu software updates is that the knowledge of updates enables a local eavesdropper to build a list of packages and their versions that are installed on a machine. If some of the packages are outdated and have known vulnerabilities, an adversary might use it as an attack vector. A percentage of software updates still occurs over un-encrypted connections, which is still an issue; but encrypted connections to software-update repositories also expose which distribution and the kind of update being done (security / restricted[1] / multiverse[2] / etc). We hope that this unnecessary leakage will disappear in the near future.

The YouTube dataset contains 191,250 unique videos, obtained by iteratively querying the YouTube API. One semantic video is generally represented by $2-5$ .webm files, which corresponds to various video qualities. Hence, each object in the dataset is a unique (video, quality) pair. We use this dataset as if the videos were downloaded in bulk rather than streamed; that is, we pad the video as a single file. The argument for padding YouTube videos as whole files is that, as shown by related work [174, 175, 189], variable-bitrate encoding combined with streaming leak which video is being watched. If YouTube wanted to protect the privacy of its users, it could re-encode everything to constant-bitrate encoding and still stream it, but then the total length of the stream would still leak information. Alternatively, it could adopt a model similar to that of the iTunes store, where videos have variable bit-rate but are bulk-downloaded; but again, the total downloaded length would leak information, requiring some padding. Hence, we explore how unique the YouTube videos are by length with and without padding.

The files dataset was constituted by collecting all file sizes in the home directories (‘`~user/`’) of 10 users, on machines running Fedora, Arch, and Mac OS X.

Finally, the Alexa dataset is made of 2,627 websites from the Alexa Top 1M list. The size of each website is the sum of all the resources loaded by the web page, which has been recorded by piloting a ‘chrome-headless’ instance with a script, mimicking real browsing. One reason for padding whole websites – as opposed to padding individual resources – is that related work in website fingerprinting showed the importance of the total downloaded size [82]. The

[1] Contains proprietary software and drivers.
[2] Contains software restricted by copyright.

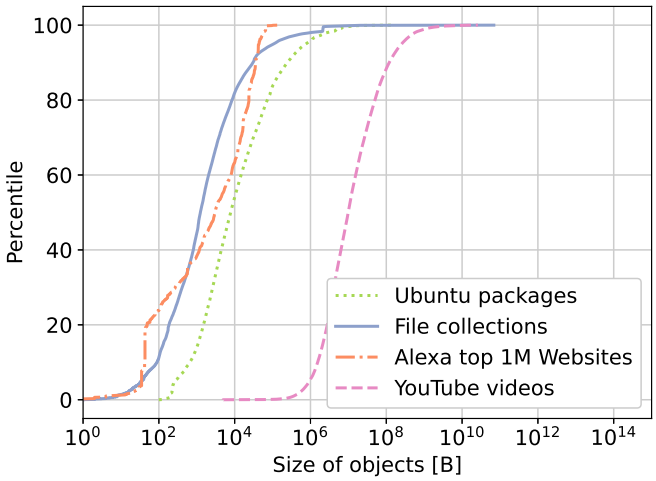


Figure 3.3 – Distribution of the sizes of the objects in each dataset.

effectiveness of Padmé when padding individual resources, or for instance bursts [228], is left as interesting future work.

3.5.2 Evaluation of Padmé

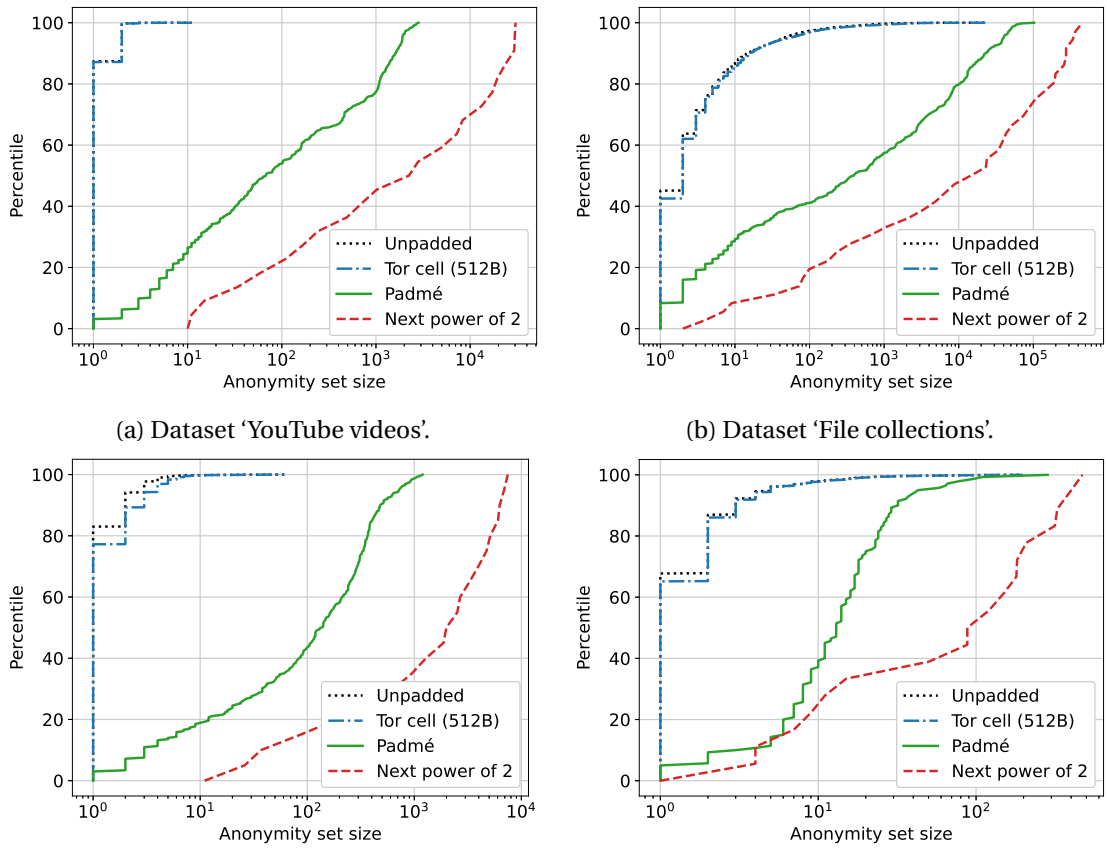
The distribution of the objects sizes for all the datasets is shown in Figure 3.3. Intuitively, it is harder for an efficient padding scheme to build groups of same-sized files when there are large objects in the dataset. Therefore, we expect the last 5% to 10% of the four datasets to remain somewhat unique, even after padding.

For each dataset, we analyze the anonymity set size of each object. To compute this metric, we group objects by their size, and report the distribution of the sizes of these groups. A large number of small groups indicate that many objects are easily identifiable. For each dataset, we compare three different approaches: the NextP2 strawman, Padmé, and padding to a fixed block size of 512 B, like a Tor cell. The anonymity metrics are shown in Figure 3.4, and the respective overheads are shown in Table 3.3.

For all these datasets, despite containing very different objects, a large percentage of objects have a unique size: 87% in the case of YouTube video (Figure 3.4a), 45% in the case of files (Figure 3.4b), 83% in the case of Ubuntu packages (Figure 3.4c), and 68% in the case of websites (Figure 3.4d). These characteristics persist in traditional block-cipher encryption (blue dashed curves) where objects are padded only to a block size. Even after being padded to 512 B, the size of a Tor cell, most object sizes remain as unique as in the unpadded case. We observe similar results when padding to 256 B, the typical block size for AES (not plotted).

NextP2 (red dotted curves) provides the best anonymity: in the YouTube and Ubuntu datasets (Figures 3.4a and 3.4c), there is no single object that remains unique with respect to its size; all belong to groups of at least 10 objects. We cannot generalize this statement, of course, as shown by the other two datasets (Figures 3.4b and 3.4d). In general, we see a massive improvement with respect to the unpadded case. Recall that this padding scheme is impractically costly, adding +100% to the size in the worst case and +50% in mean. In Table 3.3, we see that the mean overhead is of +45%.

Finally, we see the anonymity provided by Padmé (green solid curves). By design, Padmé has an acceptable maximum overhead (maximum +12% and decreasing). In three of the four



(a) Dataset 'YouTube videos'.

(b) Dataset 'File collections'.

(c) Dataset 'Ubuntu packages'.

(d) Dataset 'Alexa top 1M websites'.

Figure 3.4 – Analysis of the anonymity provided by various padding approaches: NextP2, Padmé, padding with a constant block size and no padding. NextP2 provides better anonymity, at the cost of a drastically higher overhead (at most +100% instead of +12%).

datasets, there is a constant difference between our expensive reference point NextP2 and Padmé; despite having a decreasing overhead with respect to L, unlike NextP2. This means that although larger files have proportionally less protection (*i.e.*, less padding in percentage) with Padmé, this is not critical, as these files are rarer and are harder to protect efficiently, even with a naïve and costly approach. When we observe the percentage of uniquely identifiable objects (objects that trivially reveal their plaintext given our perfect adversary), we see a significant drop by using Padmé: from 83% to 3% for the Ubuntu dataset, from 87% to 3% for the YouTube dataset, from 45% to 8% for the files dataset and from 68% to 6% for the Alexa dataset. In Table 3.3, we see that the mean overhead of Padmé is around 3%, more than an order of magnitude smaller than NextP2. We also see how using a fixed block size can yield high overhead in percentage, in addition to insufficient protection.

Table 3.3 – Analysis of the overhead, in percentage, of various padding approaches. In the first column, we use b =512 B as block size.

Dataset	Fixed block size	Next power of two	Padmé
YouTube videos	0.01	44.12	2.23
File collections	40.15	44.18	3.64
Ubuntu packages	14.09	43.21	3.12
Alexa top 1M websites	36.71	47.12	3.07

3.6 Related Work

The closest related work PURBs build on is Broadcast Encryption [27, 36, 74, 86, 96], which formalizes the security notion behind a ciphertext for multiple recipients. In particular, the most relevant notion in (Private) Broadcast Encryption is Recipient Privacy [27], in which an adversary cannot tell whether a public key is a valid recipient for a given ciphertext. PURBs goes further by enabling multiple simultaneous suites, while achieving indistinguishably from random bits in the ind$-cca2 model. PURBs also addresses size leakage.

Traffic morphing [237] is a method for hiding the traffic of a specific application by masking it as traffic of another application and imitating the corresponding packet distribution. The tools built upon this method can be standalone [224] or use the concept of Tor pluggable transport [157, 229, 232] that is applied to preventing Tor traffic from being identified and censored [209]. There are two fundamental differences with PURBs. First, PURBs focus on a single unit of data; we do not yet explore the question of the time distribution of multiple PURBs. Second, traffic-morphing systems, in most cases, try to mimic a specific transport and sometimes are designed to only hide the traffic of one given the other. PURBs are indistinguishable from random string regardless of the application.

Traffic-analysis aims at inferring the contents of encrypted communication by analyzing metadata. The most well-studied application of it is website fingerprinting [82, 164, 226, 227], but it has also been applied to video identification [174, 175, 189] and VoIP traffic [46, 236]. In website fingerprinting over Tor, research has repeatedly showed that the total website size is the feature that helps an adversary the most [52, 82, 163]. In particular, Dyer et al. [82] show the necessity of padding the whole website, as opposed to individual packets, to prevent an adversary from identifying a website by its observed total size. Wang et al. [228] propose deterministic and randomized padding strategies tailored for padding Tor traffic against a perfect attacker, which inspired the design of Padmé.

Finally, Sphinx [68] is an encrypted packet format for mix networks with the goal of minimizing the information revealed to the adversary. Sphinx shares similarities with PURBs in its binary format (*e.g.*, the presence of a group element followed by a ciphertext). Unlike PURBs, however, it supports only one cipher suite, and one direct recipient. To the best of our knowledge, PURBs is the first solution that hides all metadata while providing cryptographic agility.

3.7 Limitations

Padmé is a generic approach which we observed to provide good trade-off in some domains (§3.5) while having better theoretical properties than trivial solutions. Our design is a generic approach that is likely to be outperformed by solutions tailored for their application domain.

A practical limitation behind the concept of PURBs is when attempting to decrypt a ciphertext for which the recipient does not possess the recipient key. In this case, the recipient will attempt potentially a high number of trial-decryptions (logarithmic in the size of the ciphertext). This can be a limitation in scenarios in which minimizing CPU computations is important (*e.g.*, a high-volume web server performing TLS decryptions). To avoid leaking timing information, a variant of this limitation also applies to a recipient with a decryption key: the decryption implementation must behave as if his key is was the last possible position in the PURB. These constraints comes from the design choice that the header and the payload cannot be told apart without a decryption key.

3.8 Conclusion

We presented a way to reduce metadata leakage from an encrypted file or message, while at the same time maintaining cryptographic agility for the recipient.

Conventional encrypted data formats leak information, via both unencrypted metadata and ciphertext length, that may be used by attackers to infer sensitive information via techniques such as traffic analysis and website fingerprinting. We have argued that this metadata leakage is not necessary, and as evidence have presented PURBs, a generic approach for designing encrypted data formats that do not leak anything at all, except for the padded length of the ciphertexts, to anyone without the decryption keys. Despite having no cleartext header, PURBs can be efficiently encoded and decoded, and can simultaneously support multiple public keys and cipher suites. Finally, we have introduced Padmé, a padding scheme that reduces the length leakage of ciphertexts and has a modest overhead decreasing with file size. In terms of "object anonymity", Padmé performs better than classic padding schemes with fixed block size, and its overhead is asymptotically lower than using exponentially increasing padding.

4. Reducing Metadata Leakage from Communications

In this chapter, we focus on building Anonymous Communication Systems (ACNs). These systems enable users to communicate (*e.g.*, among themselves, or with the Internet) while reducing the metadata that is exposed compared to their non-anonymous equivalent. We propose two new systems: (1) PriFi, a system that behaves like a VPN but that strongly protects user identities, and (2) Rubato, a metadata-private messaging system for mobile devices.

4.1 Introduction

Traditional communication systems (*e.g.*, emails, instant messengers) can harm the privacy of their users by revealing sensitive metadata, such as the identity of the people communicating or the time a particular user is sending messages.

Anonymous Communication Networks (ACNs) are systems built to hide some of this metadata; in particular, the identities of the communicating parties, and the times at which a particular honest user is sending messages. Many ACNs were developed in the past 20 years of research (§4.3). Few of these systems, however, have seen a widespread use. One of the reasons might be technical: such systems have tremendous running costs compared to their non-anonymous equivalent (as discussed briefly in §4.2). This hypothesis is supported by the fact that the most widespread ACN is Tor, a system that does not protect all metadata for the sake of efficiency [210], although researchers have since then proposed more secure and costly alternatives.

Hence, one of our goals in this thesis is to propose new systems that protect communication metadata. The contributions we make in this chapter remove some of the limitations of the state of the art in ACNs, thus facilitating low-latency communications, or enabling mobile users to participate. However, our proposed systems do not solve the overarching problem of metadata leakage in communications, which we believe is a difficult problem that cannot be fixed at a reasonable cost yet.

Contributions in this Chapter. We address two limitations of ACNs.

One limitation is that most ACNs have latency that is too high for "low-latency" categories of traffic, such as VoIP or video streaming. The ACNs that do support these categories of traffic are often tailored exclusively for these types of traffic [142, 144, 145]. This creates friction for users, who might find useful a single system that protects communication metadata while they browse the Web, write emails, and perform a voice call (similar to a VPN). We present PriFi (§4.4), an ACN that provides low-latency communications in the context of a local-area network. The protocol hides the source of a message by ensuring that the communication

patterns of all participants are equal, even in the presence of active attacks. PriFi has a high bandwidth cost, but ensures low-latency, traffic-agnostic communication for a small set of users.

Another limitation of many ACNs is *synchronicity* (or *sender coordination*): users follow a global schedule that divides the time in rounds, in which all users must participate [20, 26, 141, 142, 220]. This requirement might be acceptable for non-battery-constrained devices that are always online. However, mobile devices might sporadically disconnect to save battery energy or due to network conditions; they cannot be expected to reliably participate following a global fixed schedule, especially when the rounds are short. We present Rubato (§4.5), an ACN for instant messaging that has a latency in seconds to minutes and handles millions of users. Rubato is a circuit-based mixnet that enables its users to be asynchronous; that is to say, to participate in the network according to their own schedule.

4.2 Background

This section is a short highlight of some common concepts in the field of anonymous communications. The relevant notions are explained more formally in the following sections (§4.4 and §4.5).

Model and Adversary. ACNs consider some users who want to communicate securely and anonymously. Whereas the former goal is achieved through traditional end-to-end encryption and authentication, ACNs are also concerned with keeping private the sender/receiver identities against an adversary A. Most works consider an adversary who observes all network messages, who controls some of the clients and servers, and who is able to actively tamper with messages. Such a strong adversary is believed to be a realistic threat [125].

The security notion of "anonymity" encompasses several more fine-grained notions: *e.g.,* unobservability (whether a sender/receiver communicates or not), or unlinkability (between a sender and a receiver, between a sender/receiver and a message). These notions are not mutually exclusive; one can imply the other, as formalized by Kuhn *et al.* [134].

In this thesis, we propose two systems that target two different notions:

1. In PriFi (§4.4), where anonymous messages can be sent to a non-anonymous recipient outside of the system, we seek unlinkability between an honest sender Alice and any message m output by the protocol: an adversary cannot infer whether Alice sent m (written Alice$\rightarrow_m$) or not (written Alice$\not\rightarrow_m$).
2. In Rubato (§4.5), the security notion provides unlinkability between an honest sender Alice and an honest recipient Bob: it hides whether they are in communication (written Alice$\leftrightarrow$Bob) or not (written Alice$\not\leftrightarrow$Bob).

In both cases, we can formalize the security notion in terms of the adversary's observations $\mathcal{O}$. Taking the example of Rubato (the case of PriFi is similar), we write

$$\Pr[\mathcal{O}|\text{Alice} \leftrightarrow \text{Bob}] \approx \Pr[\mathcal{O}|\text{Alice} \not\leftrightarrow \text{Bob}] \tag{4.1}$$

Informally, the adversary's observations $\mathcal{O}$ does not change significantly whether Alice and Bob communicate or not. The probabilities are taken over the random processes and over the random coins used in the system.

Most systems do not conceal that Alice uses the ACN in the first place. The probability that she will communicate with Bob must be independent from when she uses the system, which the adversary can observe. If Alice does not communicate with anyone, most systems require her to send dummy traffic (also called *chaff*) to fool the adversary.

Levels of Anonymity. In practice, Expression 4.1 can take different forms. Some works, including PriFi (§4.4), provide perfect anonymity (under a cryptographic assumption): in this case, Expression 4.1 has an equality sign in the middle. This means that the adversary's observations are identical whether Alice and Bob communicate or not.

Some related work, including Rubato (§4.5), provide a weaker, differential-privacy guarantee [81], often with the benefit of increased performance. In this case, the expression is rewritten as:

$$\Pr[\mathcal{O}|\text{Alice} \leftrightarrow \text{Bob}] \leq e^{\epsilon} \Pr[\mathcal{O}|\text{Alice} \not\leftrightarrow \text{Bob}] + \delta, \text{ and}$$
$$\Pr[\mathcal{O}|\text{Alice} \not\leftrightarrow \text{Bob}] \leq e^{\epsilon} \Pr[\mathcal{O}|\text{Alice} \leftrightarrow \text{Bob}] + \delta$$

for small values of $\epsilon, \delta \geq 0$. Intuitively, this enables the adversary to learn a small amount of statistical information, bounded by ϵ and δ.

Traffic Analysis and Adversarial View. The analysis of the communication metadata of ACNs is also called *traffic analysis*; and an ACN that protects or does not leak metadata is said to be *traffic-analysis resistant*.

Some older approaches are not traffic-analysis resistant [67, 78, 91, 176]: the analysis of communication metadata (*e.g.,* packet timings) can reveal the sender or receiver. These systems consider either a weaker form of anonymity, or an adversary that cannot observe such metadata (*e.g.,* because he only observes some portion of the network). In the latter case, Expression 4.1 could still hold, but $\mathcal{O}$ does not consist of all network features. Recent systems typically strive to resist traffic analysis against a global passive adversary, or even a global active adversary (see §4.3).

On the high cost of ACNs. We briefly highlight the reason why ACNs are inherently costly. We consider an ACN as a black box that receives and forwards messages. Suppose that at some point in time, Alice sends a message to Bob through the ACN. The adversary A observes a message leaving Alice's computer; he might also observe other messages sent by other users. We want to hide the fact that Alice is a sender; intuitively, we want any honest user to be the potential sender. Then, it is necessary that *all* honest users also transmit a message before the system delivers the message to Bob. Each idle participant is trivially excluded from the list of potential senders.

Therefore, to achieve sender anonymity, an ACN system can follow (a combination of) two paradigms:

1. Wait upon all participants to have a real message to send before delivering any message; this introduces delays.
2. Have all users send dummy traffic at regular intervals, even when they have no real message to send; this adds bandwidth usage.

This results in the necessity to spend either significant time (through delays) or bandwidth to reduce communication metadata. In contrast, traditional systems that do not protect metadata

can be more efficient. Often, users can send messages without delays or any synchronization with other users; when they have nothing to send, users can simply be idle.

This intuition has been formalized as the Anonymity Trilemma by Das *et al.* [70].

4.3 Related Work

Anonymous communications are often built upon one of the following primitives: onion-routing [100] and mix-networks [47], DC-nets [48], or on variants of Private Information Retrieval (PIR) [53]. Both onion-routing networks and mix-networks (*mixnets*) forward messages over a chain of servers, called *routers* or *mix-servers*. At each server, messages are re-encrypted and potentially delayed. The security guarantee lies in the difficulty for an eavesdropper to learn the whole path taken by a message. Onion-routing systems (and some mix networks) often build *circuits* that are reused by many packets [78]; these systems typically do not add artificial delays and focus on providing low-latency. Mixnets tend to delay and batch messages, and to have synchronous rounds.

Onion-routing. Tor [78] has the largest user-base. It operates at the application layer and establishes circuits that are reused by many packets. To provide low-latency communications, messages are not delayed. Packets have a fixed size, but no end-to-end padding is used. Hence, Tor does not provide traffic-analysis resistance against a global passive adversary, and many attacks have been proposed to break its anonymity guarantees [160, 164, 168, 225, 226].

A category of work target better performance by operating on the network layer. LAP [119] and Dovetail [187] provide "lightweight" anonymity against a local adversary; they re-encrypt packet headers but not the contents. Hornet [49] considers a stronger adversary that observes multiple network location, but that does not fully observe the network. Finally, TARANET [50] uses circuits that have constant traffic even in the presence of active attacks, achieving traffic-analysis resistance and low-latency. Network-layer solutions are agnostic to the transported contents, which can be a disadvantage when designing padding.

Some work target the peer-to-peer setting: Crowds [176] targets Web browsing and pick paths on the network using a random walk; it has been shown to be vulnerable to route capture attacks (*i.e.,* malicious nodes proxying all traffic among themselves) and predecessor attacks [238]. MorphMix [177] presents a collusion-detection technique to avoid the route capture attack. Tarzan [91] instead makes collusion among nodes harder by selecting paths using pseudo-randomness; it however requires a lookup mechanism to keep track of all nodes in the network. I2P [243] is a fully decentralized low-latency network that provides its own closed-world services, but does not resist a global adversary [212]. ACNs in the P2P setting scale better than their counterparts and remove single points of failure [243]. However, the node discovery and the selection of an unbiased path are challenging in this setting.

Mixnets. Mixnets have been introduced by Chaum [47]. It was followed by several proposals to mix emails: Babel [111], Mixmaster [158] (a *type II remailer*), Mixminion [67] (a *type III remailer*). These systems collect messages until a threshold is reached, at which point they are sent to the next hop; this introduces significant latency. The latter design by Danezis *et al.* use a new packet format to handle replay attacks and tagging attacks. They improve resilience against flooding attacks using a timed dynamic pool batching strategy, yet do not fully resist such active attacks. Several other mixing strategies have been presented, notably Stop-and-Go mixes [129]. In such systems, messages are independently delayed at mixes without batching,

and the mix work in a continuous manner. More complex designs eliminate some active attacks using techniques such as zero-knowledge proofs [122] and verifiable shuffles [93, 159].

Modern mix-network approaches improve scalability and efficiency with new topologies and other techniques. Vuvuzela [220] is a mixnet with differential-privacy guarantee against a strong, active attacker. The differential-privacy guarantee enables quantifying precisely what is revealed to an adversary performing an active attack, and enables Vuvuzela to precisely chose how much noise should be injected in the system to resist the attack. Alpenhorn [143] is a variant optimized for the bootstrapping of communication: users don't communicate directly on Alpenhorn but derive a key that can be used with other systems. Alpenhorn's dialing mechanism uses Identity-Based-Encryption to provide forward-secret key exchanges without using a global key directory. Stadium [216] improves upon Vuvuzela and tackles horizontal scalability. Its mixnet is layered: it has (1) small groups of anytrust servers that mix a fraction of the messages, and (2) distribution layers that mix messages between all anytrust groups. It uses verifiable processing to ensure that noise messages are properly forwarded through the mixnet. In the same vein, Atom [136] focuses on horizontal scalability using a mixnet composed of small anytrust groups of servers. Atom is a free-path mixnet that uses a new re-encryption scheme. Unlike Stadium and Vuvuzela, Atom relies on standard cryptographic assumptions and not on differential privacy: thus, it has stronger guarantees against active attacks at the cost of worse performance. However, to ensure that messages can be exchanged, users must send more data compared to Atom. Karaoke [141] improves upon Vuvuzela with a new topology, and by reducing the amount of noise and computations needed. Messages in Karaoke either double-pair with a recipient, or pair with themselves in the absence of communication. This reduces the information revealed to the adversary. The noise verification is done using Bloom filters instead of expensive proofs. The topology is also more horizontal than in Vuvuzela, and Karaoke uses longer paths to achieve proper overall mixing. Yodel [142] builds upon Karaoke and focuses on low-latency for VoIP calls. It uses fast circuits that only require symmetric cryptography, and that are re-used for several messages during a conversation. Each user has up to one active conversation at any time.

Aqua [145] is a system to provide anonymous file sharing. It uses a weaker threat model than related works: Users are organized in trust zones, which corresponds to their anonymity set, and their provider in this trust zone is assumed to be honest. Herd [144] is an anonymous communication system for VoIP with similar assumptions. To reduce bandwidth usage, Herd proposes a hybrid architecture with a first layer of P2P clients, which connect to a set of servers on the Internet. Both these works do not resist a global adversary who observes the source and destination trust zone.

Loopix [171] is a modern Stop-and-go mixnet that delays messages independently. Loopix uses Poisson noise and exponential delays to ensure that mixes are *memoryless*, and hence achieve good mixing without batching. Mix-servers and users use self-loops to detect some active attacks. Users connect to a semi-trusted service provider that sends and receives messages for them if they disconnect. AsynchroMix [151] uses an asynchronous multiparty computation to mix broadcast messages. It focuses on shuffling messages when servers can go offline or delay processing messages. Finally, XRD [138] proposes a novel design composed of parallel independent mix-chains. Each mix-chain thwarts active attacks using aggregate hybrid shuffles. To communicate across chains and achieve global mixing, each users participates in l of these mix-chains, where l is minimized while ensuring that every pair of users have at least one mix-chain in common. Hydra [188] uses circuits to connect two endpoints like Tor's hidden services. It addresses some of Tor's vulnerabilities by routing messages through a

mixnet on synchronous rounds. However, it does not hide the number of active conversations to the adversary, which allows for intersection attacks.

PIR-based. Riposte [61] enables anonymous microblogging using a "reverse-PIR" that allows private writes, similar to a DC-net. They improve communication efficiency using distributed point functions and using a database design that can recover from two-ways collisions. They propose two systems: an efficient one based on two non-colluding servers, and a more heavyweight variant that uses a set of anytrust servers. Riffle [137] is a more efficient variant that uses a mixnet for uploading messages and a PIR scheme for download. In a similar vein to Yodel, they use a hybrid construction that first sets up a shuffle using asymmetric cryptography, but then anonymizes messages using only fast symmetric cryptographic operations. However, their guarantee is cryptographic, not differentially-private. Like DC-nets, Riffle requires re-running the setup phase in case of client churn.

Pung [8] proposes a novel PIR construction to lower the costs of anonymous communications. Pung optimizes the bandwidth costs using a private search to first locate indices of interest, then by batching the retrievals. However, despite the improvements brought by the amortized operations, fetching a message still involves a number of operations that is linear in the size of the database. Express [84] optimizes a PIR construction to reduce the cost of reads, while making writes more expensive. As a result, clients in Express can retrieve messages cheaply but the system only supports sending tens of messages per second across all clients.

Metal [51] provides anonymous file sharing using two semi-honest, non-colluding servers. It uses a combination of secure two-party computation and ORAM to allow users share these files, even when they are not online at the same time.

DC-nets. DC-nets [48] work by having all clients send a share of a message. These shares are collected and combined into one anonymous message. By design, users are synchronized and must send the share at the appropriate time: the collective output is delayed until all shares are sent. Most systems have provable traffic-analysis resistance and typically have high bandwidth and latency costs.

Dissent [62, 233] and Verdict [63] operate at the application layer; Dissent brings DC-nets to the anytrust model with a client/server reducing the number of keys, whereas Verdict thwarts some active attacks with a verifiable DC-net. Herbivore [99] propose to organize users in small clusters to improve communication efficiency. Finally, DC-nets have been further studied with an emphasis on detecting malicious behavior from insiders [63, 101], collisions resolutions [94, 95], user scheduling [133] and applications to voting [219].

4.4 PriFi: Anonymous Communications for Local-Area Networks

We now present PriFi, an ACN tailored for organizational networks: it protects users from eavesdropping and tracking attacks. PriFi targets the setting of local-area networks (*e.g.*, non-governmental organizations, companies, universities), for which typical ACNs tailored to the Internet and are poorly suited. PriFi exploits the characteristics of LANs and WLANs to provide high-performance traffic-analysis resistant communication against a strong adversary.

PriFi builds on Dining Cryptographers networks (DC-nets), but reduces the high communication latency of prior designs via a new client/relay/server architecture, in which a client's packets remain on their usual network path without additional hops, and in which a set of remote servers assist the anonymization process without adding latency. PriFi also solves the

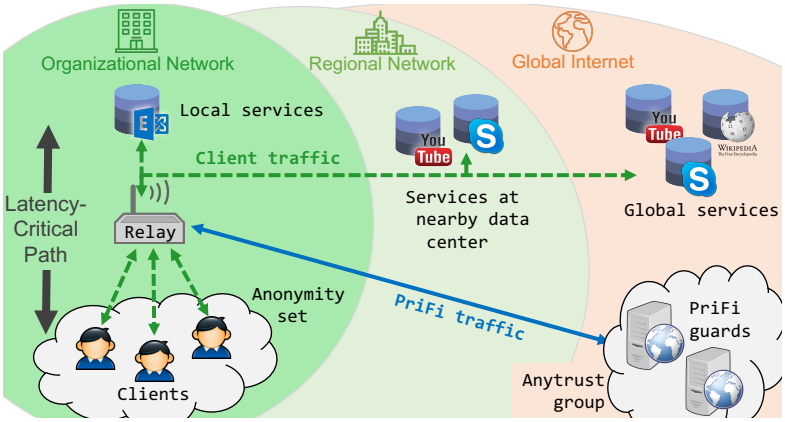


Figure 4.4.1 – PriFi's architecture consisting of clients, a relay, and group of anytrust servers called the *guards*. Clients' packets remain on their usual network path without additional hops, unlike in mix-networks and onion-routing protocols.

challenge of equivocation attacks, which are not addressed by related work, by encrypting traffic based on communication history. Our evaluation shows that PriFi introduces modest latency overhead (≈ 100ms for 100 clients) and is compatible with delay-sensitive applications such as Voice-over-IP.

Acknowledgements. This work has been made possible by the help of Dr. Italo Dacosta, Dr. Ennan Zhai, Dr. Mahdi Zamani, Dr. Apostolos Pyrgelis. We also thank Julien Weber, Pierre Sarton, Chen Jun and Carlos Villa Sanchez for their help with software engineering.

4.4.1 Introduction

Local-area networks (LANs and WLANs) deployed in organizational networks are vulnerable to eavesdropping attacks. Sensitive traffic is usually encrypted, but metadata such as *who is communicating* and the communication patterns remain visible. Such metadata enable an eavesdropper to identify and track users passively [90, 240], to infer contents and endpoints [42, 46, 80, 103, 164, 189, 226, 231], and potentially to perform targeted attacks on high-value devices and users. Eavesdropping attacks can be performed by a single compromised endpoint or malicious user. This is particularly worrisome when the users are loosely trusted, or when the organizational network is deployed in an adverse environment. For example, the International Committee of the Red Cross (ICRC) has strong privacy and security needs regarding their communications: a previous study confirms that its "staff and beneficiaries need to communicate in a multitude of adverse environments that are often susceptible to eavesdropping, to physical attacks on the infrastructure, and to coercion of the personnel" [146].

To protect against eavesdropping in LANs, few solutions exist today. Standard encryption protocols (*e.g.,* TLS, VPNs, IPsec) do not hide the identity of the communicating entities [109, 110, 179]. VLANs isolate users but do not hide the traffic patterns on wireless channels, which remains visible even when the network infrastructure is not compromised, *e.g.,* in the case of a parking-lot attack[1]. In both cases, an attacker can use traffic-analysis attacks to passively track individuals and infer communication's contents (*e.g.,* identifying endpoints such as

[1] Attacking or eavesdropping on a wireless network from the parking lot outside a company building, *e.g.,* using specialized antennas.

websites [42, 103, 164, 226], contents such as videos streams [80, 189], or even English words in encrypted Skype calls [46, 231]).

Anonymous communication networks (ACNs) are designed to conceal the communicating entities; however, most ACNs are designed for the Internet and translate poorly to the LAN setting. Most ACNs rely on mix-networks or onion-routing, a common drawback of which is that their security relies on routing the traffic through a series of servers distributed around the Internet [49, 50, 78, 136, 171]. First, this implies that internal communications in the organization's network would need to be routed over the Internet. More importantly, to minimize the risks of coercion and collusion, these servers are typically spread across different jurisdictions, hence these designs introduce significant latency overhead.

Dining Cryptographers networks (DC-nets) [48] are an anonymization primitive that could be attractive in terms of latency in some contexts, as their security relies on information coding and not on sequential operations done by different servers. In theory, therefore, anonymity can be achieved without high-latency server-to-server communication. This theoretical appeal has not been achieved in practice, however. Previous DC-net systems such as Dissent [62], Dissent in Numbers [233], and Verdict [63] still use costly server-to-server communication, thus imposing latencies in the order of seconds [233], and notably routing users' traffic through the servers.

We present PriFi, the first low-latency anonymous communication network tailored to organizational networks. PriFi provides anonymity against global eavesdroppers: Users are assured that their communication patterns are indistinguishable from the communications of other PriFi users, even if the local network infrastructure is compromised. To anonymize IP packet flows, PriFi works at the network level like a VPN. Unlike a VPN, however, PriFi's security does not depend on a single endpoint, and the protocol provably resists traffic analysis. PriFi provides low-latency, traffic-agnostic communication suitable for delay-sensitive applications such as streaming and VoIP, at the cost of higher LAN bandwidth usage.

Compared with previous work, PriFi significantly reduces communication latency through a new three-tier architecture composed of *clients*, a *relay* in the LAN (*e.g.,* a router), and *guard* servers that are geographically distributed over the Internet (Figure 4.4.1). This architecture is compatible with organizational networks, and enables PriFi to avoid major latency overheads present in other ACNs. Unlike previous DC-net systems that use multi-hop, multi-round protocols, and costly server-to-server communications [62, 63, 233], PriFi achieves similar guarantees while removing all server-to-server communications from the latency-critical path. To produce anonymous output, PriFi ciphertexts pre-computed and sent by the guards are combined locally at the relay, so that relay↔guard delay does not affect the latency experienced by clients. Moreover, the traffic from clients remains on its usual network path, client↔relay↔destination, and does not go through the guards. Some added latency results from buffering and software processing, but not from additional network hops. As a result, PriFi's latency is 2 orders of magnitude lower than the closest related work with the same setup [233].

We also present a solution for *equivocation attacks*, *i.e.,* de-anonymization by a malicious relay sending different information to different clients and analyzing their subsequent behavior. Previous DC-net systems are vulnerable to this attack, but do not address it [62, 63, 233]. Equivocation attacks can be detected using consensus or gossiping between the clients, at a high bandwidth and latency cost. We present a new low-latency, low-bandwidth solution that relies on binding encryption to communication history.

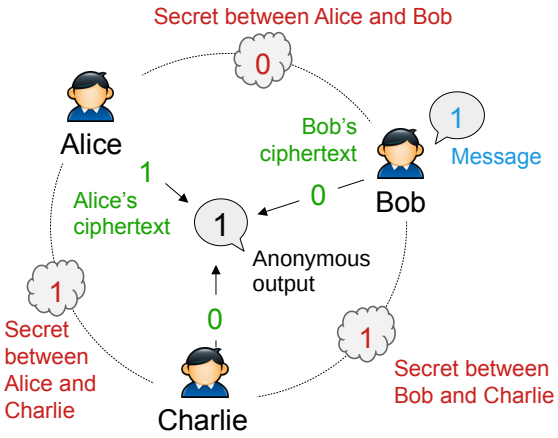


Figure 4.4.2 – Example of a 1-bit DC-net.

We evaluate PriFi on a topology corresponding to an organizational network. We observe that the latency overhead caused by PriFi is low enough for VoIP and video conferencing (≈ 100 ms for 100 users), and that the internal and external bandwidth usage of PriFi is acceptable in an organizational network (≈ 40 Mbps in a 100 Mbps LAN). In comparison, the latency of the closest related work, Dissent in Numbers [233], is 14.5 seconds for 100 clients on the same setup. One part of the evaluation is dedicated to the ICRC scenario; we replay real network traces recorded at an ICRC delegation and find that the increase in latency is tolerable in practice (between 20 and 140ms on average).

Our contributions in this section are as follows:

- PriFi, a low-latency, traffic-agnostic, traffic-analysis-resistant anonymous communication network, building on a new DC-nets architecture optimized for LANs;
- A low-latency method of protecting DC-nets against disruption attacks (*i.e.*, jamming) by malicious insiders;
- A new low-latency defense against equivocation attacks;
- An analysis of the effect of user mobility on DC-nets.

4.4.2 Background

A Dining Cryptographers network or DC-net [48] is a protocol that provides anonymous broadcast for a group of users who communicate in lock-step, in successive rounds. In a given round, each user produces a ciphertext of the same length. One user also embeds a plaintext in its ciphertext. Combining all users' ciphertexts reveals the plaintext, without revealing which user sent it.

Figure 4.4.2 shows an example of a 1-bit DC-net. Each pair of users derives a shared secret (in red). Each user's ciphertext is the XOR of his shared secrets (in green). Bob, the anonymous sender, also XORs in its message (in blue). By XORing all ciphertexts together, all shared secrets cancel out, revealing the anonymized message. If at least two users are honest, this protocol achieves unconditional sender anonymity. Every user sends a ciphertext of the same length, ciphertexts from honest parties are indistinguishable from each other without all shared secrets, and a single missing ciphertext (from an honest party) prevents the computation of the output.

In practice, to produce ciphertexts for multiple rounds, shared secrets are used to seed pseudo-random generators.

Impact of Topology. The original DC-net design [48] requires key material between every pair of members. Dissent [233] and subsequent work [63] present a more scalable two-tier topology made of clients and servers, which reduces the number of keys. However, the client/server topology has a significant negative impact on latency: It requires several server-to-server rounds of communication to ensure integrity, accountability, and to handle churn. PriFi avoids this drawback with a new client/relay/guard architecture.

Disruption Protection. Vanilla DC-nets are vulnerable to disruption attacks from malicious insiders [48], where a malicious user can corrupt other clients' messages. Some previous work uses proactively verifiable constructions that are too slow for low-latency communication [63]. Others use "trap bits" and "blame mechanisms" [234] that require minutes to hours to find disruptors, mainly due to expensive server-to-server communication. PriFi uses a new retroactive blame mechanism to detect disruptors in seconds.

Equivocation Protection. Answers to anonymous messages are typically broadcast to all users. Previous DC-net designs did not address equivocation attacks, where a malicious server sends each client different, identifiable information to distinguish the anonymous receiver (see §4.4.7). Equivocation thus represents a practical attack vector against prior systems [62, 63, 233]. Equivocation attacks can be detected using consensus [139] or gossiping [194] between the clients, at a high bandwidth and latency cost. In PriFi, in contrast, messages from clients cannot be decrypted if an equivocation attack occurs, and PriFi protects against this threat without communication among clients.

4.4.3 System Overview

PriFi is similar to a low-latency relay or gateway service within a LAN, like a VPN or SOCKS proxy, which tunnels traffic between clients and the relay (*e.g.,* a LAN router). Informally, these tunnels protect honest clients' traffic from eavesdropping attacks. The traffic is anonymized, preventing a third party from assigning a packet or flow to a specific device or end-user. Additionally, unlike traditional proxy services, (1) the relay need not be trusted, *i.e.*, security properties hold in case of compromise, and (2) the communications provably resist traffic-analysis attacks.

4.4.3.1 System Model

Consider n clients $C_1, \ldots, C_n$ that are part of an organizational network and are connected to a *relay* R. The relay is the gateway that connects the LAN to the Internet (*e.g.,* a LAN or WLAN router, Figure 4.4.1) and typically is already part of the existing infrastructure. The relay can process regular network traffic, in addition to running the PriFi software.

The system uses a small set $S_1, \ldots, S_m$ of m servers, called *guards*, whose role is to assist the relay in the anonymization process. These guards could be maintained by independent third parties, similar to Tor's volunteer relays, or sold as a "privacy service" by companies. To maximize diversity and collective trustworthiness, these guards are distributed around the world, preferably across different jurisdictions. Therefore, the connections between the guards and the relay are assumed to have a high latency.

4.4.3.2 Threat Model

Let A be a computationally-bounded global passive adversary who observes all network traffic. In addition to the passive adversary, as PriFi is a closed-membership system, we consider and address active attacks from insiders, but not active attacks from outsiders, which are fairly orthogonal to PriFi and can be addressed via adequate server provisioning [172] or denial-of-service protection [166].

Most clients may be controlled by the adversary A, but we require at least two honest clients at all times: otherwise, de-anonymization is trivial. The guards are in the *anytrust* model [62, 220, 233]: we assume that least one guard is honest, but a client does not need to know which one, and we assume all guards to be highly available.

Finally, the relay is considered *malicious but available*: it may actively try to de-anonymize honest users or perform arbitrary attacks, but it will not perform actions that only affect the availability of PriFi communications such as delaying, corrupting, or dropping messages.

4.4.3.3 Goals

Anonymity. In PriFi, users can send messages to entities outside of the LAN. Similarly, a sender outside of the LAN may send a message to a PriFi member (it is then broadcasted to all members). A sender or recipient external to a PriFi network is not anonymous. Thus, we model Alice and any message m output by the protocol, without specifying a recipient: Alice might be the sender of m (Alice $\rightarrow_m$) or not (Alice $\not\rightarrow_m$). Given the adversary's A observations $\mathcal{O}$, the formulation for anonymity in PriFi is:

$$\Pr[\mathcal{O}|\text{Alice} \rightarrow_m] = \Pr[\mathcal{O}|\text{Alice} \not\rightarrow_m] \tag{4.2}$$

That is, the adversary's observation do not change whether Alice is the sender of a message or not. The formulation for Alice receiving a message is identical.

The definition is tied to a particular message m because for the sake of efficiency, the goal is not to hide all metadata: the adversary is allowed to observe whether any user is sending/receiving a message or not – but not which honest user. PriFi protects one honest user's traffic among all honest users' traffic, but does not hide global/aggregate communication volumes or time-series of packets. Informally, an eavesdropper could learn that some honest user is browsing the Web, or using VoIP, but not which honest user.

Accountability. We also informally specify a second goal: Misbehaving parties should be traceable without affecting the anonymity of honest users.

4.4.3.4 PriFi Solution Overview

PriFi starts with a setup phase where clients authenticate themselves to the relay. Clients and guards then derive shared secrets. Finally, clients are organized in a *schedule* (a secret permutation) to decide when they communicate.

Upstream Traffic. The clients and the guards run a DC-net protocol. Communication occurs in short *time slots*. In each time slot, each client and guard sends a ciphertext to the relay. The *slot owner* can additionally embed some payload. The relay waits for all ciphertexts, then computes the anonymized output. This reveals one or more IP packet(s) without source

address; the relay replaces it with its own IP address (as in a NAT) and forwards it to its destination.

Due to the construction of the DC-net, this protocol ensures provable anonymity. Ciphertexts are indistinguishable from each other to the adversary. During a slot, each client sends exactly the same number of bits. Finally, if the contribution from any honest client is missing, the output is undecipherable.

Precomputation of Ciphertexts. The ciphertexts from the guards are independent of the anonymous payloads. Hence, a key optimization is that guards' ciphertexts are batch computed and sent in advance to the relay. The relay buffers and pre-combines the ciphertexts from multiple guards, storing a single stream of pseudo-random bits to be later combined with clients' ciphertexts. This enables PriFi to have low latency despite the presence of high-latency links between the guards and the relay.

Latency-Critical Path. Another important advantage of combining locally the ciphertexts is that clients' packets remain on their usual network path. The added latency is due mostly to the relay's need to wait for all clients. Similar systems that route clients' traffic between servers distributed around the Internet incur much higher latency.

Downstream Traffic. When receiving an answer to an anonymous message sent in some time slot, the relay encrypts it under the (anonymous) slot owner's public key, then broadcasts the ciphertext to all clients. As each client receives exactly the same message, this achieves anonymity goal for downstream traffic.

For broadcasting the downstream message, rather than performing n unicast transmissions, the relay exploits the LAN topology and uses UDP broadcast, letting layer-2 network equipment (*e.g.,* switches) replicate the message if needed. In WLANs, such a broadcast requires only one message, achieving receiver anonymity at no bandwidth or energy cost in the absence of link-layer retransmissions.

4.4.4 Design

4.4.4.1 Preliminaries.

For convenience, all symbols used are summarised in Appendix B, Table B.1.

Let λ be a standard security parameter, and let $\mathbb{G}$ be a cyclic finite group of prime order where the Decisional Diffie-Hellman (DDH) assumption [35] holds (*e.g.,* an elliptic curve).

Let $(\texttt{KeyGen}, \mathcal{S}, \mathcal{V})$ be a signature scheme, with $\texttt{KeyGen}(\mathbb{G}, 1^\lambda)$ an algorithm that generates the private-public key pair (p, P) usable for signing. We denote as $\mathcal{S}_p(m)$ the signature of the message m with the key p.

Let $\texttt{KDF} : \mathbb{G}(1^\lambda) \rightarrow \{0,1\}^\lambda$ be a key derivation function that converts a group element into a bit string that can be used as a symmetric key. Let $(\mathcal{E}, \mathcal{D})$ be a ind$-cca2-secure symmetric nonce-based encryption scheme [183]. We denote as $\mathcal{E}_k(m)$ the encryption of the message m with the key k.

Let $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a standard cryptographic hash function. We use the random oracle model to model the output of H. Let $\texttt{PRG} : \{0,1\}^\lambda \rightarrow \{0,1\}^*$ be a standard pseudo-random generator. Let $F_1 : \{0,1\}^\lambda \rightarrow \mathbb{G}$ be a public, invertible mapping function from binary strings to $\mathbb{G}$, and let $F_2 : \{0,1\}^* \rightarrow \mathbb{G}$ be a hash function to $\mathbb{G}$ which maps to any point in $\mathbb{G}$ with uniform

probability (*e.g.,* Elligator Squared [211]). Finally, let $F_3 : \{0,1\}^* \to \mathbb{N}$ be a public function that maps bitstrings to non-zero integers.

Identities. Each party has a long-term key pair (denoted with the *hat* symbol) generated with $\texttt{KeyGen}(\mathbb{G}, 1^\lambda)$:

- $(\hat{p}_i, \hat{P}_i)$ for client C_i, with $i \in \{1, \dots n\}$
- $(\hat{p}_j, \hat{P}_j)$ for guard S_j, with $i \in \{1, \dots m\}$
- $(\hat{p}_r, \hat{P}_r)$ for the relay

Let v be the vector notation for v. For each epoch, the group definition G consists of all long-term public keys $G = (\hat{P}_i, \hat{P}_j, \hat{P}_r),\ i \in \{1, \dots n\},\ j \in \{1, \dots m\}$, and G is known to all parties (*e.g.,* via a public-key infrastructure). Finally, let $T = (\hat{P}_1, \hat{P}_2, \dots)$ be a static roster of allowed clients known to the relay and the clients (*e.g.,* via a configuration file).

4.4.4.2 Protocols

PriFi starts with the protocol `Setup` (Algorithm 4.1), followed by several runs of the protocol `Anonymize` (Algorithm 4.2).

Setup. Each client authenticates itself to the relay using its long-term public key, and generates a fresh ephemeral key pair. Then, each client C_i runs an authenticated Diffie-Hellman key exchange protocol with each guard S_j, using the fresh key pair to agree on a shared secret r_{ij}. This secret is used later to compute the DC-net's ciphertexts.

Then, to produce a permutation π, the guards shuffle the client's ephemeral public keys P_i by using a verifiable shuffle (*e.g.,* Neff's verifiable shuffle [159]). The public keys in π correspond to the keys in P_i, in a shuffled order, such that no one knows the full permutation. Only a client holding the private key p_i corresponding to an input in P_i can recognize the corresponding pseudonym key in π.

Anonymize. After `Setup`, all nodes continuously run `Anonymize`. In each time slot, clients and guards participate in a DC-net protocol. All guards compute one ℓ-bit pseudo-random message from the `PRG`s seeded with the shared secrets and send it to the relay. All clients perform likewise, except for the client owning the time slot, who additionally includes its upstream message(s) m_i in the computation. In practice, m_i is one or more IP packet(s) without source address, up to a total length ℓ. If the slot owner has nothing to transmit, it sets $m_i = 0^\ell$.

Once the relay receives the $n + m$ ciphertexts from all clients and guards, it XORs them together to obtain m_k. If the protocol is executed correctly, m_k is equal to m_i, as the values of $\texttt{PRG}(r_{ij}), i \in \{1 \dots, n\}, j \in \{1 \dots, m\}$ cancel out. If m_k is a full IP packet, the relay replaces the null source IP in the header by its own (just like in a NAT) and forwards it to its destination. If it is a partial packet, the relay buffers it and completes it during the next schedule.

Then, the relay broadcasts one downstream message d to all clients, each $d \in d$ being encrypted with a public key $\tilde{P}_k \in \pi$ corresponding to an anonymous client. We emphasize that the relay does not know for which client it encrypts. Additionally, d is of arbitrary length ℓ', possibly much larger than ℓ, easily accommodating downstream-intensive scenarios. Finally, we emphasize that d can contain data for multiple users (from previous rounds). If the relay has nothing to transmit, it sends a single 0 bit to indicate the end of the round.

Algorithm 4.1: `Setup`

Inputs: $\lambda, \mathbb{G}, G, T$

Outputs: schedule π, shared secrets r_{ij} between each pair of client/guard $(C_i,\ S_j)$

1. Client→Relay Auth. Each client C_i generates a fresh key pair $(p_i, P_i) \leftarrow \texttt{KeyGen}(\mathbb{G}, 1^\lambda)$ and sends $P_i, \mathcal{S}_{\hat{p}_i}(P_i)$ to the relay. The relay checks the signature and that $\hat{P}_i \in T$, and replies with $\mathcal{S}_{\hat{p}_r}(P_i)$.

2. Client→Guard Auth. Each client C_i sends $P_i, \mathcal{S}_{\hat{p}_r}(P_i), \mathcal{S}_{\hat{p}_i}(P_i)$ to all guards.

3. Shared Secrets. Each client C_i derives m secrets $r_{ij} = \texttt{KDF}(p_i \cdot \hat{P}_j)$; similarly, each guard S_j derives n secrets $r_{ij} = \texttt{KDF}(\hat{p}_j \cdot P_i)$ for each client with a valid signature $\mathcal{S}_{\hat{p}_r}(P_i)$ from the relay.

4. Verifiable Shuffle. Clients participate in a verifiable shuffle protocol [159] run by the guards, with the ephemeral keys P_i as input. The public output π consists of n pseudonym keys in permuted order, such that no one knows which client corresponds to which key except the owner of the corresponding private key. More formally, we write $\pi = (\tilde{P}_{\alpha(1)}, \ldots, \tilde{P}_{\alpha(n)})$, where $\tilde{P}_{\alpha(i)} = c \cdot P_i$ for a permutation α and some constant c. At the end of this step, clients receive π along with a transcript signed by all guards.

Safety checks. In step 4, the honest guard checks that each input P_i corresponds to a client with a valid $\mathcal{S}_{\hat{p}_r}(P_i)$, or aborts.

At the end of `Setup`, honest clients check that (1) the verifiable shuffle completed correctly, (2) π is signed by every guard in G, (3) there are at least $K = 2$ clients in T in the input, and (4) that its own shuffled pseudonym is included in the permutation; if any test fails, it aborts.

Finally, the relay creates n empty dictionaries b_k, indexed by $k = \alpha(i)$, to keep track of IP sockets later used for packet forwarding.

Algorithm 4.2: `Anonymize`

Inputs: r_{ij}, π, upstream message size ℓ, downstream message size ℓ'

Outputs: per round k: anonymous message m_k, downstream traffic d.

For round $k \in \{1, \ldots n\}$:

- Each client C_i sends to the relay $c_i \leftarrow \texttt{DCNet-Gen}(r_{ij}, x_i)$ with
  $$x_i = \begin{cases} m_i, & \text{if } \alpha(i) = k, \\ 0, & \text{otherwise.} \end{cases} \qquad \text{// } C_i \text{ is the sender}$$
- Each guard S_j sends to the relay $s_j \leftarrow \texttt{DCNet-Gen}(r_{ij}, 0)$.
- The relay computes $m_k \leftarrow \texttt{DCNet-Reveal}(c_i, s_j)$, with $m_k \in \{0,1\}^\ell$ an IP message.
- The relay handles m_k as follows:
  - If m_k does not correspond to an active socket in b_k, the relay dials the connection and stores the socket.
  - it puts its own IP address in m_k, then sends it in the appropriate socket in b_k. // NAT
- The relay computes $d \leftarrow \texttt{Downstream}(b)$ and sends d to each client. // broadcast

Function `DCNet-Gen`(r_{ij}, x_i)**:**
  return $\bigoplus_{r \in r_{ij}} \texttt{PRG}(r) \oplus x_i$ // XOR of all pseudo-random pads and $x_i = m_i$ or 0

Function `DCNet-Reveal`(c_i, s_j)**:**
  return $\bigoplus_i c_i \oplus \bigoplus_j s_j$ // XOR all values

Function `Downstream`(b)**:**
  $d \leftarrow$ array();
  for $k \in \{1, \ldots n\}$ **do**
    for socket b_k containing downstream bytes d, add $\mathcal{E}_{\tilde{P}_k}(d)$ to d // poll connections, encrypt
  end

4.4.4.3 Limitations of this protocol

Accountability. No mechanism enforces dishonest parties to correctly follow the protocol; malicious parties can anonymously disrupt the communications. This is a well-known DC-net issue [48, 62, 233], addressed in Section 4.4.6.

Downstream Anonymity. This notion refers to the clients being indistinguishable when receiving downstream messages, which is trivially the case if the relay truthfully sends the same downstream data to all clients. This property is not enforced above, but is addressed later in Section 4.4.7.

4.4.5 Practical Considerations

End-to-End Confidentiality. A malicious relay can see the upstream message plaintexts. This is also the case for a VPN server or Tor exit relay; clients should use standard end-to-end encryption (*e.g.,* TLS) on top of PriFi.

Churn. In the case of churn, *e.g.,* if any client or guard joins or disconnects, the relay broadcasts a `Setup` request that signals the start of a new epoch. Upon reception, each node aborts and re-runs `Setup`. Churn negatively affects performance; we evaluate its effect in Appendix B.6.

Bandwidth Usage. To reduce idle bandwidth usage, the relay periodically sends a "load request" in which clients can anonymously *open* or *close* their slots. The relay skips a closed slot, hence saving time and bandwidth. If all slots of a schedule are closed, the relay sleeps for a predetermined interval, further saving bandwidth at the cost of higher initial latency when resuming communications. The concrete parameters of this improvement (*e.g.,* frequency of the load requests, sleep time) are not fully explored in this work; they exhibit a typical latency-bandwidth usage trade-off.

We emphasize that load tuning does not reduce the anonymity set size; all clients still transmit ciphertexts exactly at the same time. Load tuning makes global communication volumes and packet timings more visible to an external eavesdropper, but our threat model considers a stronger, local eavesdropper (the malicious relay) who has access to this information anyway.

Finally, although this has not been investigated in this work, both up/down-stream sizes ℓ and ℓ' can be dynamically tuned without interrupting the communications. This allows the relay to better match the sending/receiving rates of the clients and further reduce idle bandwidth usage.

Synchronicity. The protocol uses message reception events, rather than clocks, to keep the participants in lock-step.

4.4.6 Disruption Protection

In the basic protocol above, a malicious active insider can modify or jam upstream communications by transmitting arbitrary incorrect bits instead of the ciphertext defined by the protocol. This is particularly problematic because the attacker is provably anonymous and untraceable.

In the related work, these attacks can be detected retroactively using "trap bit" protocols [233] that detect a disruptor with a certain probability but reduce the throughput linearly with

respect to the number of trap bits. Unfortunately, the probability of detection must be high enough to detect a single bit-flip, which can effectively corrupt a message. Another technique is to rely on commitments before every DC-net message [62], which adds latency.

Some DC-nets use group arithmetic instead of binary strings, which enables proving (proactively or retroactively) that computations are correct [63, 101]. These designs do not fit low-latency requirements, unfortunately, as their computation time is significantly higher: tens of milliseconds per message for the computation alone, whereas XOR-based DC-nets take microseconds. A brief analysis of this computational cost is provided in Verdict [63] (Figure 6).

PriFi uses a retroactive, hash-based "blame" mechanism on top of a classic XOR-based DC-net, which (1) keeps the typical operation (in the absence of jamming) as fast as possible, and reduces the bandwidth lost due to the protection, and (2) excludes a disruptor with high probability (1/2 per flipped bit). Exploiting the LAN topology, our exclusion mechanism takes seconds, which is orders of magnitude faster than the related work [233].

Protocol. In `Anonymize-With-Integrity` (Appendix B, Algorithm B.1), we modify the previous `Anonymize` protocol (Algorithm 4.2) to protect against disruption from malicious insiders.

In short, we add a hash-based detection of corruption and a blame mechanism to exclude a disruptor. With each downstream message, the relay sends the hash of the previous upstream message. If the anonymous sender detects an incorrect hash, it requests a copy of its own disrupted message by setting a flag $b_{\text{echo\_last}}$ to 1.

When receiving a disrupted copy m'_k of a previously-sent message m_k, the client searches for a bit position z such that $(m_k)_z = 0$ and $(m'_k)_z = 1$. If such z exists, then the client requests to de-anonymize the z^{th} bit of his own slot k, by sending a $\text{NIZKPoK}_{k,z}(\tilde{p}_k : \tilde{p}_k = \log \tilde{P}_k)$ in its next upstream message, a non-interactive proof of knowledge of the key $\tilde{p}_{(k \bmod n)}$ corresponding to slot k in π [34]. The proof is bound to the public values k and z. For simplicity, we write $\text{PoK}_{k,z}(\tilde{p}_k : \tilde{p}_k = \log \tilde{P}_k)$ hereafter, thus ignoring (1) the mod n and (2) the acronym for "Non-Interactive, Zero-Knowledge".

If no such z exists, the message was disrupted but the disruptor cannot be traced without simultaneously de-anonymizing a client (see "Remarks" below for more details). In this case, nothing happens.

Upon reception of the message $\text{PoK}_{k,z}(\tilde{p}_k : \tilde{p}_k = \log \tilde{P}_k)$, the relay starts the `Blame` (Algorithm 4.3) protocol, which iteratively asks all parties to prove their computations to be correct. The protocol identifies a pair of entities whose computations do not match, and ultimately excludes the misbehaving entity.

Limitations. The detection relies on the capacity of the jammed client to transmit $b_{\text{echo\_last}}$ and the PoK; the adversary also can jam these values. In practice, l is fairly large (*e.g.*, 5 kB), and the client can use redundancy coding over his message to make the task harder for the adversary. When this probabilistic solution is insufficient, users can use a verifiable DC-net [63] to transmit without the risk of jamming; in practice, this verifiable DC-net would run in background with very low bandwidth and high latency, just enough to transmit the proof-of-knowledge.

Remarks. In step 3 of `Blame` (Protocol 4.3), we observe why the client starting the blame checks that $(m_k)_z = 0$: otherwise, revealing $\texttt{PRG}(r_{ij})_z$ over a non-anonymous channel would flag this client as the sender, as $\bigoplus_j \texttt{PRG}(r_{ij})_z \neq (m_k)_z$.

Algorithm 4.3: `Blame`

Inputs: $\text{PoK}_{k,z}(\tilde{p}_k : \tilde{p}_k = \log \tilde{P}_k), c_i, s_j$

1. The relay broadcasts $\text{PoK}_{k,z}(\tilde{p}_k : \tilde{p}_k = \log \tilde{P}_k)$ to every client/guard.
2. Each client/guard checks the PoK, and reveals the z^{th} bit of the values $\texttt{PRG}(r_{ij})$ for slot k, $\forall i \in \{1,\dots n\}, j \in \{1,\dots m\}$ by sending a non-anonymous, signed message $\texttt{PRG}(r_{ij})_z, \mathcal{S}_{\hat{p}_i}(\texttt{PRG}(r_{ij})_z)$ to the relay. A non-complying entity is identified as the disruptor.
3. For each client, the relay checks the signature, and that $\bigoplus_j \texttt{PRG}(r_{ij})_z$ indeed equals $(c_i)_z$ sent in slot k; if a mismatch is detected, this client is identified as the disruptor. The relay performs the same verification for each guard.
4. For each pair of client-guard (C_i, S_j), the relay compares $\texttt{PRG}(r_{ij})_z$ from the client and $\texttt{PRG}(r_{ij})_z$ from the guard — they should be equal. If there is a mismatch, at least one of them is lying. The relay continues by forwarding the signed message $\texttt{PRG}(r_{ij})_z$ from C_i to S_j and vice-versa.
5. C_i checks that $\texttt{PRG}(r_{ij})_z$ is signed by S_j, and that the bit $\texttt{PRG}(r_{ij})_z$ is in contradiction with its own bit $\texttt{PRG}(r_{ij})_z$. Then, he answers with r_{ij}, the secret shared with S_j, along with a proof of correctness for computing r_{ij}. S_j proceeds similarly. A non-complying entity is identified as the disruptor.
6. The relay checks the proofs of correctness, then uses r_{ij} to recompute the correct value for $\texttt{PRG}(r_{ij})$ for slot k, identifying the disruptor.

Once the disruptor identified, the relay excludes it from the group G and the roster T, and then broadcasts all inputs and messages exchanged in `Blame` to all clients for accountability.

In step 5 of `Blame` (Protocol 4.3), we remark that at least one mismatching pair exists, otherwise the slot would not have been disrupted. If multiple disruptors exist, `Blame` excludes one disruptor per disruption event.

4.4.7 Equivocation Protection

In both versions of `Anonymize` (Algorithm 4.2 and Algorithm B.1), the relay broadcasts the downstream traffic d to all clients to ensure receiver anonymity. However, no mechanism enforces truthful broadcast, so a malicious relay can perform *equivocation attacks*: *i.e.,* send different messages to each client, hoping that their subsequent behaviors will reveal which client actually decrypted the message. This attack can be seen as a "poisoning" of downstream traffic.

Equivocation Example. Assume clients C_1 and C_2 are both honest. On the first round, the relay decodes an anonymous DNS request. Instead of broadcasting the same DNS answer to C_1 and C_2, the relay sends two different answers containing IP_1 and IP_2, respectively. Later, the relay decodes an anonymous IP packet with IP_2 as destination. It can guess that C_2 made the request, as C_1 has never received IP_2.

In practice, a credible scenario for such an attack is a router infected with malware or compromised by the adversary and spying on honest users of a corporate network, possibly colluding with the endpoints contacted by the clients.

We note that previous DC-net systems do not mention this issue [62, 63, 233]. The attack is possible because a malicious party relays the information, which can happen in both Dissent [233] and Verdict [63]. If the traffic is unencrypted, the attack is trivial. The use of higher-level encryption (*e.g.,* TLS) can offer a mitigation, *if* we further assume that the

remote endpoint does not collude with the malicious relay. In practice, having two particular entities under the control of the adversary (the relay and some interesting external service, *e.g.,* WikiLeaks) does not seem impossible, however.

On the contrary, we note that due to their different design, mix-nets and onion-router networks are typically not affected by this attack.

PriFi Solution. Intuitively, to thwart an equivocation attack, clients need to agree on what they have received before transmitting their next message. In PriFi, this is achieved without adding extra latency and without synchronization between clients. Clients encrypt their messages before anonymizing them; the encryption key depends on the history of downstream messages and also on the shared secrets with the guards. The relay is thus unable to recover a plaintext if not all clients share the same history.

4.4.7.1 Protocol

We modify the previous `Anonymize` and `Blame` protocols into their final versions (Algorithm 4.4 and Algorithm B.2).

`Anonymize`. Each client C_i keeps a personal history h_i of downstream communications. Upon receiving a downstream message d, each client updates the history $h_i \leftarrow H(h_i||d)$. If $F_2(h_i) = 0$, that is the representation of h_i in $\mathbb{G}$ is zero, each client performs additional hashing steps $h_i \leftarrow H(h_i)$ until $F_2(h_i) \neq 0$.

Each upstream message is then symmetrically-encrypted with a fresh key γ. This value γ is sent to the relay, blinded by a function of the downstream history h_i; if all clients agree on the value h_i, the relay can unblind γ and decrypt the message.

`Blame`. The previous `Blame` protocol finds a disruptor when values c_i or s_j are not computed correctly; we now add a way to validate the new values κ_i and σ_j. Since they are group elements, we apply a standard non-interactive zero-knowledge proof [34]. More precisely, we use an Or/And type of NIZKPoK which allows the clients to prove either (1) their ownership of the slot or (2) that κ_i is computed correctly.

4.4.8 Practical Considerations

Packet Losses. We note that the equivocation protection is decoupled from link-layer retransmissions; if one client fails to receive a packet, it will ask the relay again after a timeout, delaying all clients for this specific round (which is unavoidable for traffic-analysis resistance) but not invalidating the whole epoch with a wrong h_i value.

Remark. Due to the "Or" format of the PoK, we note that a malicious client can jam their own slot, then send arbitrary Q_i values in step 7 of `Blame`, and still pass the PoK because of their knowledge of $\tilde{p}_k$. However, this has no benefit for the adversary, as mismatching Q_i's are simply checked in the following steps of `Blame`, with no effect on honest parties.

Algorithm 4.4: `Anonymize-Final`
The differences with `Anonymize-With-Integrity` (Algorithm B.1) are highlighted in blue.

Inputs: r_{ij}, π, ℓ, ℓ'

Outputs: per round k: m_k, d.

For round $k \in \{1, \dots n\}$:

- Each client C_i sends to the relay $c_i, \kappa_i \leftarrow$ `DCNet-Gen-Client`(r_{ij}, m_i, h_i)
- Each guard S_j sends to the relay $s_j, \sigma_j \leftarrow$ `DCNet-Gen-Guard`(r_{ij})
- The relay computes $m_k \leftarrow$ `DCNet-Reveal2`$(c_i, s_j, \kappa_i, \sigma_j)$ with $m_k \in \{0,1\}^l$ or $\perp$.
- The relay handles m_k as follows:
  - if m_k is a Blame message, it starts the `Blame` protocol,
  - else, it sends m_k in the appropriate socket in b_k.
- The relay outputs $d \leftarrow$ `Downstream2`$(b, H(m_k), k, b_{\text{echo\_last}})$
- The relay updates $h_r \leftarrow H(h_r || d)$, and sends $d, \mathcal{S}_{\hat{p}_r}(h_r)$ to each client.
- When receiving d, each client checks the signature $\mathcal{S}_{\hat{p}_r}(h_r)$ or aborts.
  Then, each client C_i updates $h_i \leftarrow H(h_i || d)$. If $F_2(h_i) = 0, h_i \leftarrow H(h_i)$.

Function `DCNet-Gen-Client`(r_{ij}, m_i, h_i)**:**

1. compute x_i as follows:
   - if $\alpha(i) = k$:
     // C_i is the sender. Upstream message x_i is a Blame accusation or $\mathcal{E}_\gamma(m_i)$
     - if slot k' was disrupted, $x_i \leftarrow \text{PoK}_{k,l}(\tilde{p}_k : \tilde{p}_k = \log \tilde{P}_k)$
     - else, pick $\gamma \xleftarrow{\$} \{0,1\}^\lambda$, compute $m'_i = \mathcal{E}_\gamma(m_i)$, and set $x_i \leftarrow m'_i || b_{\text{echo\_last}}$
   - else, $x_i \leftarrow 0$
2. compute $c_i \leftarrow \bigoplus_j \text{PRG}(r_{ij}) \oplus x_i$ // DC-net computation to send x_i
3. compute κ_i as follows: // equivocation-protection tag
   - if $\alpha(i) = k$: $\kappa_i \leftarrow F_1(\gamma) + F_2(h_i) \cdot \sum_{j=1}^{m} F_3(H(\text{PRG}(r_{ij})))$ // sender transmits γ blinded
   - else, $\kappa_i \leftarrow F_2(h_i) \cdot \sum_{j=1}^{m} F_3(H(\text{PRG}(r_{ij})))$

return c_i, κ_i

Function `DCNet-Gen-Guard`(r_{ij}, x_i)**:**

$s_j \leftarrow \bigoplus_i \text{PRG}(r_{ij})$

$\sigma_j \leftarrow -\sum_{i=1}^{n} F_3(H(\text{PRG}(r_{ij})))$ // equivocation-protection tag

return s_j, σ_j

Function `DCNet-Reveal2`(c_i, s_j)**:**

$m'_k \leftarrow \bigoplus_i c_i \oplus \bigoplus_j s_j$

$F_1(\gamma) \leftarrow F_2(h_r) \cdot \sum_{j=1}^{m} \sigma_j + \sum_{i=1}^{n} \kappa_i$ // the relay unblinds γ from the κ_i, σ_j if all h_i are equal

$m_k \leftarrow \mathcal{D}_\gamma(m'_k)$

return m_k

4.4.9 Security Analysis

Anonymity. Let m be the output of the protocol `Anonymize-Final` (Algorithm 4.4) on a particular round k. Then, for an honest user Alice, our security definition requires that:

$$\Pr[\mathcal{O}|\text{Alice} \rightarrow_m] = \Pr[\mathcal{O}|\text{Alice} \not\rightarrow_m] \tag{4.3}$$

That is, the adversary's observations $\mathcal{O}$ are indistinguishable whether Alice is sender of m (Alice $\rightarrow_m$) or not.

We demonstrate that PriFi meets the security definition in three consecutive steps:

1. If `Setup` terminates without failing for any of the honest participants, then the schedule is unknown to A (Property B.3.1).
2. If `Setup` terminates without failing for any of the honest participants, for any message m output by `Anonymize-Final`, then the adversary has no advantage in guessing the source of any message sent by an honest user. In short, $\Pr[\mathcal{O}|\text{Alice} \rightarrow_m] = \Pr[\mathcal{O}|\text{Alice} \not\rightarrow_m]$ (Property B.3.2).
3. If `Setup` terminates without failing for any of the honest participants, and a slot k is disrupted, resulting in the execution of `Blame-Final`; then, the additional information revealed to A through the protocol `Blame-Final` does not affect the anonymity of honest clients (Property B.3.3).

Anti-equivocation. We demonstrate the following properties regarding equivocation attacks:

1. If two honest clients received different downstream messages $d_i \neq d_j$ on round k, then nor the relay, nor A can decrypt m_k for any subsequent rounds $k' > k$ (Property B.3.4).
2. An honest relay is never accused of an equivocation attack (Property B.3.5).

Availability. We analyze the following properties:

1. If a client (or guard) sends arbitrary values instead of the values specified in the protocol, and if `Blame-Final` is started by the relay, then one malicious client (or guard) is identified as the disruptor and is excluded from subsequent communications. (Properties B.3.6, B.3.7).
2. Honest parties are never identified as disruptors (Property B.3.9).

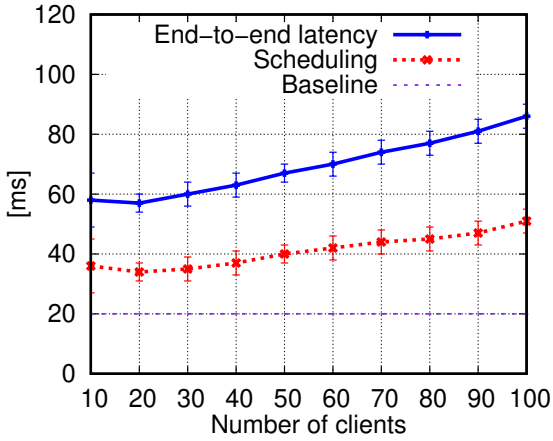
The properties and their proofs are in Appendix B.3.

4.4.10 Evaluation

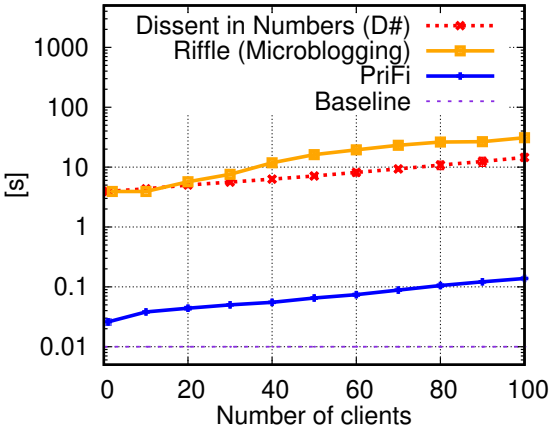
We implemented PriFi in Go [21]. We evaluate it on a LAN topology typical of a small organizational network.

Methodology. First, we measure the end-to-end latency via a SOCKS tunnel without data, by having a client randomly ping the relay. Second, we compare PriFi with the closest related work, Dissent in Numbers [233]. Third, we replay network traces that correspond to various workloads on PriFi and we measure the added latency and the bandwidth usage.

Experimental Setup. We use Deterlab [75] as a testbed. The experimental topology consists of a 100 Mbps LAN with 10 ms latency between the relay and the clients. We run three guards, each having a 10 Mbps link with 100 ms latency to the relay. We use nine machines, one dedicated to the relay and one per guard. The clients are simulated on the remaining five



(a) End-to-end latency of PriFi experienced by one client. A significant part of this latency comes from the scheduling mechanism used (red line). The baseline is twice the latency of the LAN (20 ms).



(b) Experimental comparison of the latency between PriFi and Dissent in Numbers (D#), computed as the time to send and decode an anonymous message. The baseline is the latency of the LAN (10 ms).

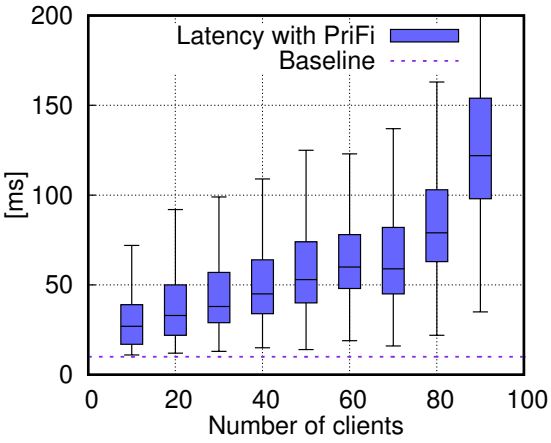
Figure 4.4.3 – End-to-end latency and comparison with Dissent in Numbers.

machines, distributed equally. All machines are 3 Ghz Xeon Dual Core with 2 GB of RAM. We focus our evaluation between 2 and 100 users, which is inspired by the ICRC's operational sites (90% of them have less than 100 users).

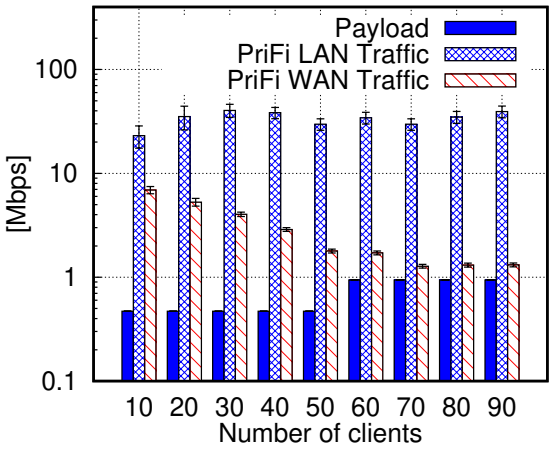
Security Parameters. We rely on the Kyber cryptographic library [73]. We use $\lambda = 256$ bits, Curve25519 for $\mathbb{G}$, SHA-256 as a hash function, and Schnorr signatures. The DC-net PRG uses BLAKE2 as an extensible output function.

Figure 4.4.3a shows the latency of the PriFi system, *i.e.,* the time needed for an anonymized packet to be sent by the client, decoded by the relay, and sent back to this same client. In this experiment, one random user is responsible for measuring these "pings", whereas others only participate in the protocol without sending data (*i.e.,* the number of active users is 1, anonymous among all users). We observe that the latency increases linearly with the number of clients, from 58 ms for 20 users (*e.g.,* a small company) to 86 ms for 100 users, and it scales reasonably well with the number of clients. We also observe that a major component of the latency is the buffering of messages by the clients; with only one slot per schedule, clients must wait for this slot before transmitting data. To reduce the time spent waiting on the slot, we alter the scheduling mechanism and let clients "close" their slot if they have nothing to send, thus enabling other clients to transmit more frequently. This reservation mechanism improves the situation where many users are idle.

Comparison with D#. We select two related works for comparison. The closest is Dissent in Numbers (abbreviated D#) [233]. Like PriFi, D# provides provable traffic-analysis by using binary-string-based DC-nets, has similar assumptions (M anytrust servers) but with no particular emphasis on being low-latency. We then compare with a more recent ACN, Riffle [137]: it has a similar topology but emphasizes on minimizing the download bandwidth for clients. We do not compare with mix-nets and onion-routing protocols, whose architecture is significantly different, in that users' messages are routed sequentially through multiple hops over the Internet. The first major difference between PriFi and both Riffle and D# is in the functionality of the guards: Both protocols require several server-to-server communications per round before outputting any anonymized data.



(a) Latency increase when using PriFi. The baseline is the latency of the LAN (10 ms).



(b) PriFi bandwidth usage. The total bandwidth usage is the sum of last two 2 bars (the payload is included in the LAN traffic). The available bandwidth in the LAN is 100 Mbps.

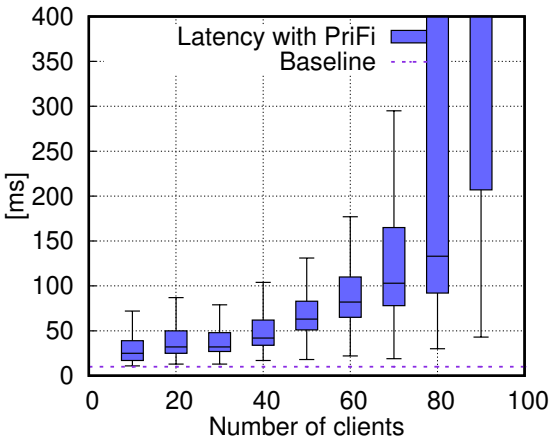
Figure 4.4.4 – PriFi performance when 5% of the users perform a Skype call (CRAWDAD dataset). The remaining 95% of the users are idle.

We deploy Riffle and D# on our setup and compare their latency against PriFi in Figure 4.4.3b. For 100 users, a round-trip message takes ≈14.5 seconds in D# (excluding setup), ≈31 seconds in Riffle (which includes a one-time setup cost of ≈7.3 seconds), and 137 ms in PriFi (excluding setup).

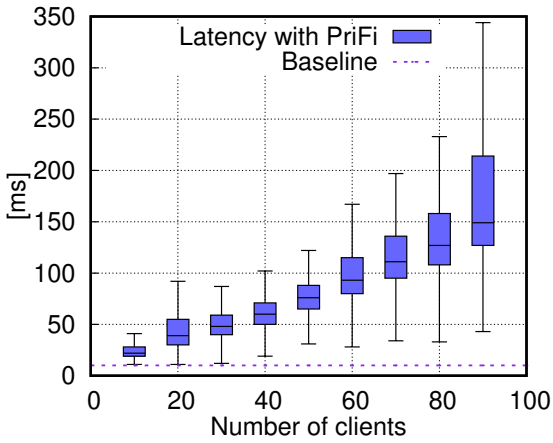
Realistic Datasets. We evaluate the performance of PriFi when replaying the dataset 'app-trafictraces' [192] from CRAWDAD [65]. We selected two sub-traces: a 'Skype' trace where one client performs a VoIP (non-video) call for 281 seconds and a 'Hangouts' where one client performs a video call for 720 seconds.

Using the same setup as before, 5% of the clients are randomly assigned packet traces from a pool and, after a random delay $r \in [0,30]$ seconds, they replay the packets through PriFi. The relay decodes the packets and records the time difference between the decoded packet and the original trace. Because most endpoints in the traces were not reachable anymore on today's Internet, the recorded latency does not include the communication to the Internet endpoints, but only the latency added by PriFi.

We also replay a dataset recorded at a ICRC delegation from June to July 2018. The capture contains network-level packet headers only, corresponding to all network traffic for 30 days of capture. During active periods, corresponding to work days, the mean number of users is 60.9, with a standard deviation of 5.8. Also during active periods, the mean bitrate of the network is of 3.1 Mbps, with a standard deviation of 4.7 and a (single) peak at 25 Mbps corresponding to a bulk file transfer. To evaluate PriFi with this dataset, we first randomly select 10 1-hour periods from the active periods (*i.e.*, we exclude weekends and nights); we replay those 10 sub-traces and measure the latency and bandwidth overhead. During this hour, we simulate a varying number of clients: First, we identify (and only simulate) local clients, identified by an IP address of the form 10.128.10.*x*; these clients replay their own packets. Second, when needed, we add additional clients who represent extra idle users. These clients send no payload data but increase the anonymity set size. We average the results over the 10 1-hour periods and over all clients.



(a) 5% of users performing a Google Hangout video call (CRAWDAD dataset). The current implementation performs well with up to 70 users.



(b) Latency overhead when replaying the 'ICRC' dataset, with 100% of users having realistic activity. This is the average over 10 1-hour period of high activity, recorded at an ICRC delegation.

Figure 4.4.5 – PriFi latency with the and 'Hangouts' (CRAWDAD) and ICRC dataset. In both cases, the bandwidth usage (not shown here) is similar to the one observed in Figure 4.4.4b (except for the payload). In both cases, the baseline is the latency of the LAN (10 ms).

Figures 4.4.4a, 4.4.5a and 4.4.5b show the added latency on the 'Skype', 'Hangouts' and ICRC datasets. Figure 4.4.4b shows the bandwidth used; PriFi has similar bandwidth usage for all three datasets.

We focus first on the latency. In Figures 4.4.4a, 4.4.5a and 4.4.5b, we observe that the latency increases with the number of clients, which is due to (1) the increasing traffic load going through PriFi, as more users send data, and (2) to the increasing time needed to collect all clients' ciphertexts. We learn the following: First, the mean added latency in the case of a Skype call (with 5% active users) is below 100ms for up to 80 clients, and below 150ms for 100 clients. The International Telecommunication Union estimates that the call quality starts degrading after a 150ms one-way latency increase [121], and users start noticing a degraded quality after a 250ms one-way latency increase [221]. Hence, the current implementation supports VoIP calls for 0–80 users and reaches its limits at around 100 clients. Second, the replay of the 'Hangouts' data exhibits similar behavior as the 'Skype' dataset; we see that the latency increases reasonably until 70 users, but then drastically increases: the current implementation cannot transmit the data fast enough and buffering occurs at the clients.

When replaying the ICRC traces, shown in Figure 4.4.5b, we observe that the added latency varies between 15 and 147 ms. This experiment was conducted with clients having network traffic corresponding to the real workload of the ICRC network. In addition, extra idle clients were simulated to achieve a constant anonymity set size of 100. This result confirms that PriFi can handle a realistic workload in the case of an ICRC delegation. We emphasize that all traffic has been anonymized through the same PriFi network, regardless of QoS. In practice, large file transfer (*e.g.,* backups) would probably either be excluded from a low-latency network, or anonymized through other means (*e.g.,* PriFi configured to provide higher throughput at higher latency).

We focus now on the bandwidth usage (Figure 4.4.4b). The bandwidth used by the system is split into two components: the bandwidth used in the LAN, and in the WAN. We also show the bitrate of the payload, as an indication of the useful throughput (goodput) of the system.

We see that the LAN bandwidth usage is typically around 40 Mbps, whereas the WAN usage varies from 6.9 to 1.3 Mbps. We recall that, in an organizational setting, the bandwidth of the LAN is typically 100 Mbps or 1 Gbps, and that the bottleneck typically lies on the link towards the Internet. We also observe that the WAN bandwidth usage decreases with the number of clients. This is a shortcoming that indicates that PriFi spends more time waiting and less time transmitting, due to the increased time needed to collect ciphers from more clients. This means that PriFi cannot fully utilize the available bandwidth to offer minimal latency. This could be mitigated by increasing the pipelining of rounds for slow clients so that all clients answer in a timely fashion.

Finally, in Appendix B.4, we briefly evaluate the limits of our implementation in terms of scalability and the impact of packet loss on the performance. Moreover, we evaluate scenarios specific to the ICRC in Appendix B.5: Our implementation supports anonymizing a remote user outside of the organization, at a latency cost of $2-4$x the baseline. When a local trusted guard can be leveraged, the latency of our protocol is reduced by a factor of 2. To conclude our evaluation, we analyze the impact of client churn in Appendix B.6.

4.4.11 Conclusion

We have presented PriFi, an anonymous communication network that protects organizational networks from eavesdropping and tracking attacks. PriFi exploits the characteristics of (W)LANs to provide low-latency, traffic-agnostic communication.

PriFi reduces the high communication latency of prior work via a new client/relay/server architecture. This new architecture removes costly server-to-server communications, and allows client's traffic to be decrypted locally, remaining on its usual network path. This avoids the latency bottleneck typically seen in other systems.

PriFi also addresses two shortcomings of the related work: First, users are protected against equivocation attacks without added latency or costly gossiping; second, leveraging the LAN topology, disruption attacks are detected retroactively and orders of magnitude faster.

We have implemented PriFi and evaluated its performance on a realistic setup, mimicking the targeted ICRC deployment. Our findings show that various workloads can be handled by PriFi, including VoIP and videoconferencing, and that restrictions usually imposed by DC-nets in case of churn when users are mobile are not problematic in PriFi.

4.4.12 Limitations

Scalability. The communication costs of our protocol scales linearly with the number of participants: $N+M$ bits of ciphertext are needed for each 1 bit of output, where N and M are the number of clients and guards, respectively. For the definition of security we target, we can demonstrate that at least N bits are necessary: exactly one contribution per client. As a result, the protocol's bandwidth usage is very high, and it would not scale to many more users.

Synchronicity. Each output message requires exactly $N+M$ messages from all participants. A delayed message from any party delays the output accordingly; a disconnection from any party implies that the output cannot be decrypted. On the side of the guards, this is partially mitigated by the fact that their contribution can be processed in advance and buffered. However, clients are in lock-step and must send messages in a synchronous and timely fashion. While our experiments showed that re-performing `Setup` is cheap, it disrupts communications for a short duration.

4.5 Rubato: Large-Scale Asynchronous Anonymous Messaging

The previous section presented an anonymous communication network that targets a small number of users in the specific setting of organizational networks. The solution provides cryptographic anonymity and has cost that is too high to scale past a few hundred users. We now focus on the opposite: achieving large-scale anonymity.

In this section, we present Rubato, a metadata-private messaging system. Rubato is the first metadata-private messaging system that supports clients on mobile devices and that scales to millions of users. It provides a differential-privacy guarantee against an attacker who controls the network, runs clients, and operates some of the system's servers. Rubato presents two technical advances that bring metadata-private messaging closer to standard large-scale mobile messaging apps. First, it supports asynchronous clients: Alice can store a message for Bob when she comes online, and Bob can later retrieve that message, without exposing the fact that he is fetching Alice's message. Second, it enables users to run clients on multiple devices with different power and bandwidth constraints and ensures safety, even if they get partitioned and cannot coordinate after an initial setup. In contrast, all prior metadata-private messaging systems that support Rubato's scale limit users to a single client that is online all the time or require each user to operate a semi-trusted server that would send and receive messages on their behalf.

We implement Rubato and test its performance by using AWS machines and a client running on a Pixel 4 smartphone. Our evaluation with 100 servers distributed across four data centers on two continents demonstrates that Rubato can achieve 32s of latency for 1M users, where each user can message up to 50 buddies in their contact list, and that Rubato can support metadata-private messaging on a mobile device. The client uses less than 300 MB/month and increases battery consumption on mobile devices by 60 mW (+19% compared to idle).

Acknowledgements. This section is based on a joint work with Moshe Kol, Dr. David Lazar, Prof. Yossi Gilad, and Prof. Nickolai Zeldovich [25].

4.5.1 Introduction

Recently, there has been significant interest in protecting metadata in large-scale messaging systems [10, 38, 138, 140, 141, 171, 216, 220], so that an adversary cannot learn which users are communicating with one another. Systems that offer linear scalability (*i.e.,* can support more users with the same performance by adding a proportional number of servers) and privacy guarantees are based on mixnets, a communication architecture where messages are shuffled to hide a sender's destination. However, mixnets offer little flexibility: all clients have to submit their messages at the same time to be mixed together. This mandatory synchronization across clients makes it difficult for mobile devices to participate in these systems for the following reasons.

First, mobile devices might go offline—*e.g.,* a user might disconnect on an airplane or because the battery of their device runs out of energy. Therefore, a messaging service for mobile devices should support *asynchronous* clients, where users can exchange messages, even if their devices are not online at the same time. Second, as mixnets are synchronous, clients have to constantly communicate with the servers in the form of cover traffic. This communication is costly in terms of cellular bandwidth and the battery life of the device. Finally, users typically communicate from multiple devices (*e.g.,* phones, tablets, and laptops). The attacker might prevent coordination between a user's devices, which is problematic if two devices contact the

same person. For example, if Alice's phone and laptop simultaneously send a message to Bob, and the adversary observes Bob receiving extra messages, the adversary can infer that Alice and Bob are *buddies* (*i.e.,* correspondents on the system).

This section presents Rubato, the first system for metadata-private text messaging on mobile devices that scales to millions of users. Rubato addresses the above challenges in supporting mobile devices, and provides a differential-privacy guarantee [81] against attackers that control the network and some of the system's servers, and that can run any number of clients. By supporting mobile devices, Rubato enables a large class of new users to participate, which increases the degree of privacy that can be provided to everyone [77, 97].

Rubato uses a hybrid design consisting of a core synchronous protocol that hides metadata based on a mixnet architecture, and an asynchronous protocol by which clients send and receive messages. The two protocols are mediated by untrusted *service providers*, whose job is to queue messages to and from each user. The service providers receive messages on behalf of different users, without learning the recipients of said messages, and without requiring chatting users to be online at the same time.

A crucial aspect of Rubato's design lies in facilitating cover traffic between buddies; this ensures that everyone receives messages at a fixed rate, regardless of when two buddies exchange messages. The challenge is to maintain a constant messaging rate when users go offline, while still enabling users to receive messages and not requiring anyone else to be trusted with the user's list of buddies. Rubato addresses this challenge by using *primed circuits*, which are reusable paths through a mixnet that enables servers to enforce fixed-rate messaging. Establishing primed circuits does not require online coordination between buddies. Alice can submit a circuit-setup message well in advance of when she will use the circuit. To communicate, Alice and Bob create primed circuits that culminate in the same dead drop, where messages are swapped. This protects the users' identities, even if one user does not manage to create their circuit.

Circuits are long-lived and created periodically in epochs, which last about a day, thus making it inexpensive to submit circuit-setup messages for several buddies and epochs in advance. Priming circuits in this manner enables Bob to send Alice messages over these circuits even if she disconnects for days, thus handling the challenge of clients going offline. But what if Alice adds a new buddy after submitting circuit-setup messages for future epochs? When Alice comes online, to support new relationships, her client can replace the circuit-setup messages queued at the service provider. To replace circuits safely, Rubato introduces a *circuit tagging* technique, which ensures that the service provider can use only one version of the circuit-setup message for each buddy. Otherwise, the provider could learn the dead drops Alice uses by submitting all setup messages and observing which dead drops correspond to more than two circuits.

Circuits enable unlinking one user who goes offline from their buddy's communication patterns. Any server along a circuit's route can fill in cover traffic if the user does not otherwise send a message to their buddy. As such, users do not need to send cover traffic at all times. Rubato also minimizes the data that a client downloads by running a fetch protocol that unlinks the messages at the service provider from the circuits they were received on; this enables the client to retrieve just a small number of messages belonging to real conversations (rather than cover traffic). Together, these techniques handle the network and power consumption challenge.

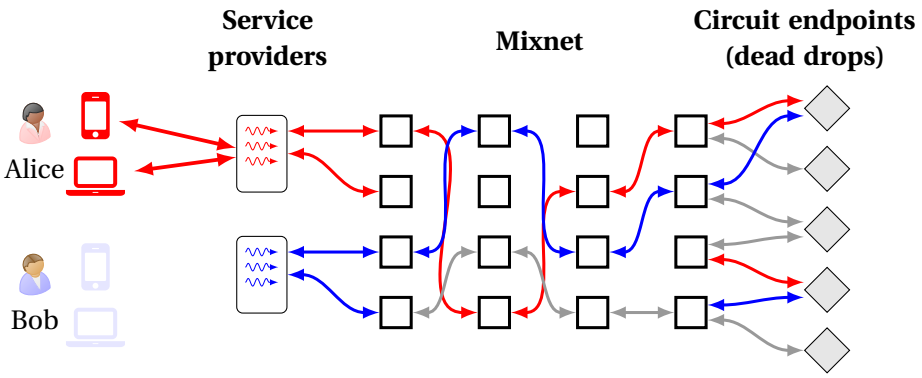


Figure 4.5.1 – In every epoch, Alice and Bob establish two primed circuits to each other. Bob is offline so his service provider submits circuit-setup messages (wavy arrows) on his behalf. The service provider also stores the user's messages (straight arrows) so its clients can catch up when coming online and can synchronize between their devices. The attacker might drop circuit-setup messages, so Rubato generates noise messages (grey arrows) to hide which dead drops correspond to real conversations.

Finally, Rubato supports users with multiple devices. It handles the challenge of partitioned devices by deterministically choosing the circuit path by using a pre-shared key across the user's devices. This ensures that, even if the devices cannot communicate with one another, they will set up circuits along the same path. Rubato's synchronous mixnet protocol then ensures that only one of these circuits will be built.

Rubato achieves differential privacy in the presence of partitioned client devices, rogue service providers, network attackers, and a portion of malicious mixnet servers. To demonstrate that Rubato's design achieves its purpose of supporting messaging for many users, we built a prototype and evaluated its performance on AWS. We also implemented a mobile client on a Pixel 4 phone and tested Rubato's applicability to the real world in terms of battery and network usage. The results show that Rubato can support 1–3 million users who send and receive messages from 50 buddies every 32–80 seconds. The client uses 220 MB–290 MB of network per month, and its battery consumption increases by 19% compared the idle phone. We recognize that Rubato's latency is high compared to typical mobile messaging apps; nonetheless, it enables large-scale metadata-private messaging for an important class of users.

In summary, the contributions of this section are the following:

- the techniques of primed circuits and circuit tagging, which allow clients to asynchronously interact with a synchronous mixnet;
- an implementation of Rubato, the first metadata-private messaging system for mobile devices that scales linearly and provides formal privacy guarantees;
- an experimental evaluation of Rubato's performance running on a mobile client.

4.5.2 Overview

Figure 4.5.1 shows an overview of communication through Rubato. It enables users to send and receive text messages from clients they run across multiple devices. Rubato's design relies on two kinds of servers: mixnet servers and service providers. Mixnet servers arrange in a full-mesh topology and operate in synchronous rounds. They shuffle messages and introduce noise at every round, which yields differential privacy about which user sent which message. The service providers serve as a bridge between this synchronous mixnet protocol and their

asynchronous clients. Each user has a designated service provider (which they choose), and the service provider's job is to queue messages, which are sent and received through the mixnet on the user's behalf, and to help the user synchronize their clients across all devices. Users do not need to trust their service provider for privacy. Therefore, they can change service providers if they suspect their provider is faulty, without worrying about exposing sensitive information to more parties (*e.g.,* if the user notices that their devices cannot synchronize or never receive messages).

Clients in Rubato set up circuits to send messages. A circuit is a connection from a service provider to a dead drop, which persists for many rounds (*e.g.,* a day's worth). The dead drop corresponds to a pseudo-random 256-bit virtual address, and the address space is shared across the mixnet servers. The circuits are routed through the mixnet, which ensures that an adversary cannot correlate the two ends of the circuit. When two circuits connect to the same dead drop, the server hosting the dead drop address swaps the messages and sends them back through the circuits; this is how users exchange messages. Messages are sent over circuits every round; these rounds are run at a relatively high frequency (*e.g.*, every minute).

4.5.2.1 Client Flexibility

Rubato's design enables asynchronous clients to send and receive messages, independently of other clients. We refer to this property as *client flexibility*. It presents a departure from a paradigm common to many previous mixnet systems [136, 138, 141, 142, 216, 220], where all clients run on the same synchronous schedule. In Rubato, clients do not need to be online at the same round to communicate and do not need to send cover traffic at every round to hide when they are not communicating. This flexibility is crucial for clients running on mobile devices that can go offline and require transmissions less frequent than a client on a desktop machine that can communicate all the time.

To achieve privacy, it is still important that the network communication pattern between a user's device and their service provider, which we call the client's *schedule*, does not reveal information about the user's communication with their buddies. The adversary can potentially infer any information that goes into deciding the client's schedule, hence it should be based only on public information. A simple but safe schedule is to connect to the service provider to send and receive messages at regular intervals (say, every few minutes). Longer intervals consume fewer resources on users' devices but at the cost of increased communication latency. Different types of devices can safely use different intervals.

Clients can change their communication pattern if the change is independent of the user's buddies. For example, Alice can stop sending and receiving messages when she boards a flight and her phone disconnects from the Internet. Clients can also piggyback on other device wake-ups (like checking for software updates) to interact with the service provider at a relatively low cost. However, changes in network patterns that depend on a user's buddies are unsafe. For instance, if Bob were to stop sending messages whenever Alice goes offline, it would signal that Bob is connected to Alice. On the contrary, it is safe for a device to miss a transmission due to a network outage, for instance.

4.5.3 Goals and Threat Model

4.5.3.1 Threat Model

Rubato aims to hide its users' communication metadata from a powerful attacker. The system considers a global active attacker who can tap all network links, and drop, inject, and modify network packets. This attacker observes, in particular, when clients connect and disconnect from the network and might partition users' devices. The attacker can also control all service providers and a portion of the system's mixnet servers and run an arbitrary number of clients. Rubato assumes that the attacker controls each mixnet server with some probability $f < 1$; this is provided as an assumption in the system's configuration. A smaller f enables Rubato to achieve better performance. An attacker could learn about a user's communication by compromising their device or the devices of their buddies [9]; we exclude such attacks from the threat model.

Rubato assumes that clients and servers know the servers' public keys, *e.g.,* through a public key infrastructure. Finally, we assume the attacker is computationally bounded and that standard cryptographic primitives, such as hash functions and encryption schemes, are secure.

4.5.3.2 Goals

Metadata Privacy. Consider an attacker and their view of the system through the network links they tap, and the service providers, mixnet servers, the clients that they operate. Rubato ensures that the attacker's view is likely to be the same whether two users, call them Alice and Bob, are buddies or not. More formally, Rubato achieves differential privacy [81]. Consider the attacker's observations $\mathcal{O}$ and the following two scenarios. In one scenario, Alice and Bob set up circuits to the same dead drop (and might exchange messages). In the other scenario, Alice and Bob set up circuits to independent dead drops (and cannot exchange messages). Rubato ensures the following inequalities for small $\epsilon, \delta \geq 0$:

$$\begin{aligned} \Pr[\mathcal{O}|\mathrm{A} \leftrightarrow \mathrm{B}] &\leq e^{\epsilon} \Pr[\mathcal{O}|\mathrm{A} \not\leftrightarrow \mathrm{B}] + \delta, \\ \Pr[\mathcal{O}|\mathrm{A} \not\leftrightarrow \mathrm{B}] &\leq e^{\epsilon} \Pr[\mathcal{O}|\mathrm{A} \leftrightarrow \mathrm{B}] + \delta \end{aligned} \tag{4.4}$$

In Equation 4.4, $\leftrightarrow$ denotes that Alice and Bob are buddies (and hence can communicate with each other), and $\not\leftrightarrow$ denotes that they are not buddies. That is, the probability for any observations the attacker could make is close under both scenarios (up to a multiplicative factor e^{ϵ} and an additive factor δ). Differential privacy quantifies the statistical information that could leak to the attacker through his observations. It provides a strong formal guarantee for the level of privacy that Rubato provides its users. Alice could plausibly deny being buddies with Bob or claim to be buddies with someone else. This privacy guarantee holds even if Alice or Bob is the single user of a rogue provider, the attacker partitions their devices, and observes the system for a long time.

Accommodating Mobile Clients. Rubato must not impose strong timing or resource requirements on clients. In particular, Rubato must allow Bob to retrieve Alice's message at any time after it reaches Bob's service provider, and accommodate clients with network and battery constraints that need to minimize communication. At the same time, Rubato must allow other clients to connect more often and achieve lower message latency.

Scale and Performance. Rubato aims to support millions of users sending and fetching messages, with latency on the order of minutes. That is, when Alice sends a message, it is

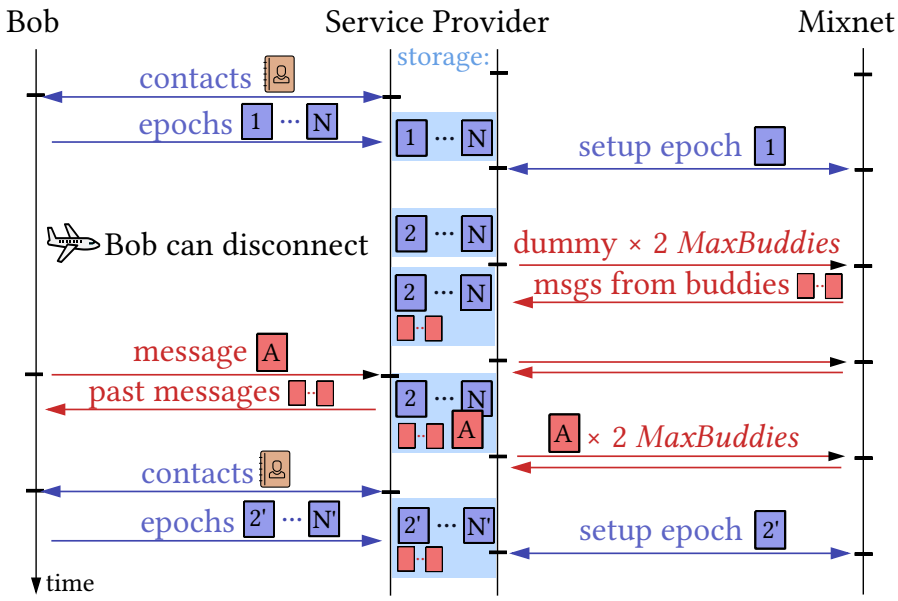


Figure 4.5.2 – Bob's client interacts with his account at the service provider. The client submits circuit-setup messages for future epochs as well as text messages for Bob's buddies. The service provider submits messages to the mixnet at the right time and stores messages for Bob so his clients can catch up when they come online.

available for Bob to retrieve about a minute later. The system must be scalable, *i.e.*, be able to provide the same performance to a larger user base by deploying more servers.

4.5.4 Design

Rubato supports users running clients on multiple devices. These devices share a secret "multi-device key" and connect to the mixnet through one service provider, as shown in Figure 4.5.1. When Alice and Bob add each other as buddies to their contact lists, their clients establish a fresh shared secret. If Alice adds Bob multiple times on her devices, perhaps because the service provider partitions them, each time their clients create a new shared secret. This secret enables Alice and Bob's clients to authenticate and encrypt messages (end-to-end) and coordinate dead drops for exchanging messages. Their clients might create this secret out-of-band (*e.g.*, by scanning QR codes if Alice and Bob meet in person) or via a metadata-private bootstrapping system such as Alpenhorn [143].

Next, we describe how users communicate through Rubato. Our description flows from the client's perspective and its interactions with the service provider. This interaction is depicted in Figure 4.5.2. Figure shows the client pseudocode and links to the respective figures. For completeness, Figure C.1 in Appendix C gives the service provider's API.

4.5.4.1 Primed circuits

Rubato splits time into epochs, which correspond to the lifetime of a circuit through which users communicate. Periodically and on a global schedule, *e.g.*, once per day, clients call `RefreshCircuits` (see Figure 4.5.3) to generate circuit-setup messages and upload them to their service providers, which submit the messages to the mixnet. The service provider queues these messages and sends them to the mixnet at the appropriate time (Figure 4.5.2), even if all of the user's clients go offline. The more epochs a client covers when calling `RefreshCircuits`,

```
while true {
   if oncePerDay {
      client.RefreshCircuits()     // Upload circuit-setup messages. Detailed in Figure 4.5.4
   }
   <-client.SendSchedule           // Block, waiting for the next wakeup event

   buddy, msg := client.OutgoingMessageQueue.Pop()
   if buddy == nil {
      msg = random.Read(msg)      // If the user has nothing to say, send cover traffic
   }
   client.SendMessage(buddy, msg) // Detailed in §4.5.4.4 and in Figure 4.5.5
   client.CheckForMessages()      // Detailed in §4.5.4.5 and in Figure 4.5.6
}
```

Figure 4.5.3 – The client's main loop. The client refreshes circuits once per day. Otherwise, it sends and receives messages according to the send schedule, which can configured to balance battery life with communication rate.

```
func (c Client) RefreshCircuits() {
   addressBook := c.serviceProvider.GetAddressBook()  // Merge address book from other devices
   prevBuddies := Decrypt(c.MultiDeviceKey, addressBook)
   c.buddies = MergeAddressBooks(prevBuddies, c.buddies)

   if len(c.buddies) < MaxBuddies {                   // Reupload padded encrypted address book
      c.buddies = append(c.buddies, GenerateFakeBuddies())
   }
   newAddressBook := Encrypt(c.MultiDeviceKey, c.buddies)
   c.serviceProvider.SetAddressBook(newAddressBook)

   for epoch := range c.serviceProvider.UpcomingEpochs() { // For the next month...
      var onions [MaxBuddies][2][]byte
      for i, buddy := range c.buddies {                    // ... upload two circuits per buddy
         rand := PRNG(c.MultiDeviceKey, i, epoch)
         onions[i][0] = GenerateCircuitSetupOnion(rand, buddy, 0)
         onions[i][1] = GenerateCircuitSetupOnion(rand, buddy, 1)
      }
      c.serviceProvider.SetCircuitSetup(epoch, onions)
   }
}
```

Figure 4.5.4 – Pseudocode for updating a client's buddy list and corresponding circuit setup onions for upcoming epochs, which are stored on the service provider. It is safe for multiple devices to run this function concurrently.

the longer the user can go offline and keep connections with their buddies. Thus, the duration of an epoch is a knob that enables trading less client communication for higher latency in adding buddies.

The pseudocode for `RefreshCircuits` is in Figure 4.5.4. First, the client synchronizes the user's contacts through the service provider, as the user might have added or removed a buddy through another device while their client was offline. The client retrieves the address book from the service provider, appends new contacts to the first available slots, and pushes the new list to the service provider for other clients. To hide when the user adds or removes a buddy, the number of slots in the address book is fixed (`MaxBuddies`). Clients pad the address book to its maximum limit and encrypt it under the multi-device key.

Second, the client primes two circuit-setup messages for each buddy and all upcoming epochs (*e.g.,* for the next month) and uploads these messages to the service provider. The reason for creating two circuits per buddy, rather than one circuit, is to enable clients to fake conversations when the user has less than `MaxBuddies`. In this case, the client creates two "idle circuits" to one dead drop (so there are exactly two circuit setup messages to all dead drops a client uses).

4.5.4.2 Noisy Circuit Establishment

Circuit Establishment. The circuit-setup message is the dead-drop address that is onion encrypted with the public keys of the servers in the route (Rubato assumes a public key infrastructure for distributing the servers' keys, see §4.5.3.1). At each layer of the onion, the client includes the next server's ID, the public half of a Diffie-Hellman handshake that allows the server to decrypt the next layer of the onion, an authentication code that ensures no preceding server has modified the onion, and the epoch number for servers to be able to ensure that circuits are established at the time the client intends. As one epoch nears the end, the mixnet runs circuit setup for the next epoch, hence circuits are ready at all times to route messages between buddies.

The servers on the circuit's route process the setup message similarly to Yodel [142]: Each server completes the Diffie-Hellman handshake, derives a symmetric key, and uses this key to verify the authentication code. After receiving the onions from all servers at the previous hop, the server de-duplicates and shuffles all circuit-setup messages, then forwards them to the next hop. The servers store the shuffle's permutation and the symmetric key for the epoch; later, they will use these records to process messages over the circuit. When the circuit-setup message reaches the dead drop, the server hosting this dead drop returns the hash of its address back through the mixnet to the user's service provider. On the route back, each server applies the inverse shuffle permutation. This hash gives an acknowledgment to the client; it means that the circuit-setup message propagated to the dead drop. Therefore, all servers in the route processed it, including any honest servers en route, which keeps their shuffle permutation secret.

Noise. Rubato cannot coordinate between clients at circuit-setup time (as users might submit setup messages for the next month and go offline). Therefore, it must ensure privacy when one client establishes a circuit and their buddy does not. Rubato mitigates leakage of sensitive information during circuit setup by noising this step with dummy circuits that its mix servers create. The dummy circuits ensure that the attacker has a similar chance of making the same observations (*e.g.*, about the network traffic and dead-drop access patterns), regardless of whether Alice and Bob are buddies. Rubato applies the message-noising technique from Karaoke [141] to circuits: For each inter-server mixnet link, each server decides a minimum number of noise circuits to route over that link by drawing from the noise distribution. Servers generate two kinds of noise: to obscure the total number of conversations, "doubles" are pairs of circuits bound to the same dead drop; to obscure the cases in which one of the buddies does not create their circuit to the shared dead drop, "singles" are lone circuits terminating at a dead drop without a pair. The servers ensure that noise circuits are established by checking that downstream servers receive them: at every hop of the circuit setup round, each server retrieves a Bloom filter from all other servers in the mixnet. A server includes all circuit-setup messages they receive in this filter. An honest server reports the correct Bloom filter, hence if noise gets dropped, its sender detects the error at the next honest server in the route. When a server sees that all its noise appears in all filters, it announces that it approves to continue. Otherwise, honest servers stop the round at the hop where noise is missing.

4.5.4.3 Safe Circuit Replacement

Priming circuits enables Rubato to support users if they go offline for a long time. However, users might add buddies after the client has submitted circuit-setup messages. Rubato enables users to update circuit-setup messages stored at the service provider, so that they can

communicate with new buddies as soon as the next epoch starts. Replacing circuits is done periodically, following a per-device schedule; if there are no changes in the buddy list, clients upload fresh circuit-setup messages pointing to the same dead drops. The key challenge in replacing circuits is that, without coordination across a user's devices, several of her clients might establish circuits to the same dead drop. This creates a distinct access pattern (*i.e.*, not the single- or double- dead drop access patterns covered by noise). An attacker controlling the dead drop's hosting server can then associate this dead drop with the user.

Rubato introduces *circuit tagging*, which enables safe replacement of old setup messages without relying on communication between a user's devices. In Figure 4.5.4, `RefreshCircuits` seeds a pseudorandom generator (`PRNG`) for each epoch with the multi-device key and the buddy's slot number in the address book. The clients include in each layer of the circuit's setup message onion a random-looking tag derived from this `PRNG`. Honest servers on the route de-duplicate circuit-setup messages according to this tag. To enable circuit de-duplication across all of a user's devices, Rubato ensures the following invariant: for each slot in the user's address book, their clients submit circuit-setup messages with the same route. With this property, even if all Alice's clients upload circuit-setup messages and a malicious service provider submits all of them, one honest server along each route suffices to observe the tag and discard duplicated circuits. Rubato achieves this invariant by decoupling the choice of the circuit's route from the dead drop at its end: The clients use the above `PRNG` to choose the same route for each buddy-slot across all devices. Although the routes are the same, each client submits different-looking messages to the service provider, because the onion encryption scheme is randomized.

4.5.4.4 Sending Messages over Circuits

The synchronous mixnet operates in rounds that compose an epoch, where users exchange messages over the circuits they set up. We envision rounds being relatively frequent, *e.g.*, on the order of every 30 seconds to a minute. On every round, a client can contact its service provider to submit and receive messages (the red flow in Figure 4.5.2), depending on their schedule. Regardless of what the clients do, each mixnet server processes precisely one message per round for every circuit that routes through it. To accommodate mobile devices, Rubato minimizes the client networking overhead.

When Alice types a message for Bob, her client pads or fragments her message to fit Rubato's fixed message-length. It encrypts and authenticates the message with the key that Alice shares with Bob and queues the message for sending. The client sends messages from this queue to the service provider by calling `SendMessage` in Figure 4.5.5. As shown in the figure, the client takes advantage of both circuits to each buddy to send two messages at a time.

The user has $2 \times$ `MaxBuddies` circuits at their provider, hence sending a message on each circuit might be costly for a mobile device with limited bandwidth. In contrast, sending a message on only one buddy's circuits reveals when circuits are active and creates correlations between chatting buddies. In Rubato, the client always sends two messages to the service provider but does not reveal which pair of circuits should route them. The service provider duplicates messages and sends them to all buddies (one on each circuit). As messages are end-to-end encrypted, only the intended buddy can decrypt them.

The servers process messages similarly to Yodel [142]. The mixnet's servers shuffle and decrypt a layer of the onion of messages arriving at them by using the permutation and symmetric keys they stored at circuit setup. Servers de-duplicate messages on the same circuit, and if a

```
func (c Client) SendMessage(buddy, msg string) {
   epoch := GetCurrentEpoch()
   round := GetNextRound()
   rd := c.serviceProvider.GetOutgoingMessage(epoch, round)

   prevMsg,prevBuddy:= Decrypt(c.MultiDeviceKey, rd.OutMsg) // Check whether another device already
   if IsRealMessage(prevMsg) {                              // queued a message; if so, re-encrypt
      msg, buddy = prevMsg, prevBuddy                       // the previous message
   } else {
      msg = MsgHeader(epoch, round, buddy) ++ msg           // Compose new message as header ++ msg
   }

   msg1, msg2 := SplitMessage(msg)                          // Split and encrypt message over the two
   ix := epoch.getBuddyIndex(buddy)                         // circuits corresponding to the buddy
   onion1:= EncryptSymOnion(epoch.circuits[ix][0].keys,msg1)
   onion2:= EncryptSymOnion(epoch.circuits[ix][1].keys,msg2)

   c.serviceProvider.SetOutgoingMessage(epoch, round, RoundData{
      Onion:  {onion1, onion2},
      OutMsg: Encrypt(c.MultiDeviceKey, {buddy, msg}),
   })
}
```

Figure 4.5.5 – Client pseudocode for sending messages. This code prevents a second device from accidentally overwriting a user-typed message with cover traffic.

circuit does not have a message, they fill-in a dummy message in its place to ensure all circuits have one message. The symmetric encryption ensures that any random message that a server fills-in is indistinguishable from a real message to any server along the route.

When the service provider has no message to send, *e.g.,* if the user is offline, it skips the user's circuits. The first honest server on the circuit's route will later fill in a dummy in its place so that dead-drop accesses will look the same.

Avoiding Overwrites. Two of the user's clients might attempt to write messages at the same mixnet round. There is no privacy risk as each circuit will only route one message per round, but one client's message might overwrite the other. Rubato ensures that clients do not overwrite each other's messages by synchronizing them through the service provider. Clients submit an encrypted bit under the multi-device key with every message that tells whether their message is cover traffic or for a buddy. Before a client submits a message, it checks with the service provider for pending messages. If there is a pending message to a buddy, the client replaces it with another encryption for the same content. Otherwise, it uploads a new onion-encrypted message.

4.5.4.5 Retrieving Messages

When a dead drop receives two messages from the mixnet, the server hosting it swaps the two messages' content and routes them back to the users' service providers through the mixnet (each mixnet server applies the inverse permutation). When a service provider receives a message from the mixnet, it stores that message for the user (see Figure 4.5.2).

Fetch Protocol. To minimize the client's bandwidth and energy costs, most messages fetched by clients should be real messages from their buddies (rather than cover traffic). Rubato's fetch protocol enables the client to retrieve a subset of the messages pending at the provider, without revealing which messages are fetched.

First, Bob's client receives from the service provider a two-byte header for each message it has for Bob. The header acts as a pseudorandom flag shared between Alice and Bob: When Alice sends a message to Bob, her client derives the header's value from the current round

```
func (c Client) CheckForMessages() ([][]byte) {
   epoch := GetCurrentEpoch()
   round := GetNextRound()
   mixers := RandomMixnetPath(length=11)

   headers := c.serviceProvider.GetHeaders(epoch, round)   // Get 2 byte per message in our inbox

   indices = []                                // Identify the indices of real messages from buddies
   for i, buddy := range c.buddies {
      if header[i] == MsgHeader(epoch, round, buddy){
         indices += i
      }
   }

   pk, sk := GenerateDHKeypair()
   nonces := c.serviceProvider.ShuffleInbox(ShuffleParams{ // Shuffle our inbox with a fresh key
      Epoch:          epoch,
      Round:          round,
      PublicKey:      pk,
      Mixers:         mixers,
   })

   shuffledIxs := PredictPositions(sk, mixers, nonces, indices)  // Predict the (shuffled) indices
   PadWithFakeRequests(shuffledIxs)

   onions := c.serviceProvider.FetchInbox(shuffledIxs) // Fetch messages at the (shuffled) indices

   msgs := DecryptOnions(onions, sk, mixers, nonces)   // Remove encryption added by ShuffleInbox
   Unshuffle(msgs, sk, mixers, nonces)              // Map the messages back to the correct buddy
   return msgs
}
```

Figure 4.5.6 – Client pseudocode for fetching messages from the service provider.

and the key Alice and Bob share, then her client sends it inside the mixnet's onion, along with the message (Figure 4.5.6). Bob's client derives the same pseudorandom headers and compares them against the headers from Alice's circuits. If they match, his client fetches the corresponding message in the second step (otherwise, it fetches a message at random). It is crucial that the service provider does not learn from which circuits Bob fetches. Rubato achieves this by mixing Bob's messages, as follows.

When the client calls `CheckForMessages` (with pseudocode in Figure 4.5.6), it triggers the provider to submit all the messages pending for Bob to a "fetch mixchain", a sequence of mix servers chosen by the client. The provider also relays to the first server in the list a fresh public Diffie-Hellman handshake key that the client chooses independently of other clients. The first mix uses its secret key to complete the Diffie-Hellman handshake and derives a shared secret with the client; it then chooses a fresh nonce and hashes it with this shared secret to derive an ephemeral symmetric key. From this ephemeral key, the server derives the shuffle permutation and symmetrically-encrypts the messages. The server passes its nonce, the client's public key, and the list of the remaining mixers to the following mix, who continues in the same fashion. The nonces ensure that even if a rogue provider replays a request, the server's outputs will look freshly random. The last server gives the shuffled messages back to the provider with the list of all servers' nonces.

Bob's client retrieves the list of nonces from the service provider. It can then derive the symmetric key for each mix chain server and compute the (shuffled) position of the messages it should fetch. Finally, the client fetches messages at the shuffled indices directly from the provider; the freshness of Diffie-Hellman key ensures that it accesses random-looking locations each time. To maintain privacy, the number of messages a client downloads must be derived from information that the attacker can safely observe. Specifically, clients download a small fixed number of messages for every round where they were offline; say, download six messages to support up to three buddies simultaneously messaging the user. Clients fetch

a different fixed number of messages to quickly catch up after being offline for an extended time, for example, fetch 500 messages per offline day.

4.5.5 Privacy Analysis

In our analysis, we first reduce the security of Rubato under partitioned devices and rogue service providers to the security of the mixnet, which also has formal treatment in prior work [141, 142]. We then analyze the differential privacy guarantee that Rubato's mixnet, coupled with the noise, provides in every epoch. The composition theorem from the differential-privacy literature enables us to compute Rubato's differential-privacy guarantee after multiple differentially-private epochs [81, 3.20], as we do in §4.5.6

4.5.5.1 Partitioned Devices

Rubato assumes that the attacker controls the network and can partition the user's devices. After circuit setup, partitioned devices might only overwrite each other's messages, which is not a privacy risk (§4.5.4.4). Therefore, we focus on circuit setup. The attacker can distribute old versions of the address book to different devices, or submit circuit-setup messages from multiple devices. The risk here is that circuits from different devices become established and create distinct dead-drop access patterns (not covered by noise). Rubato solves this problem by using circuit tagging and its mechanism for choosing routes and dead drops. Next, we show that such attacks are equivalent to dropping circuit-setup messages on the network, an attack that Rubato handles through differential privacy (as we show in §4.5.5.3).

Property 4.5.1. An attacker that partitions a user's devices can obtain the same view on the user's communication by dropping some of the user's circuit-setup messages.

Proof. Consider two partitioned devices of user Alice, d and d', and a circuit they establish for the buddy at slot i in Alice's address book. Both devices submit setup messages for circuits with the same route and tag; they derive them from the buddy's slot number, the multi-device key, and the epoch number, which are all the same for d, d' even if their view of Alice's address book differs (see Figure 4.5.4). Assume, for now, that an honest mix server exists on this circuit's route. The server will de-duplicate the two circuit-setup messages and ensure that only one circuit will be established (§4.5.4.3).

There are two cases to consider. First, the circuits from d, d' reach the same dead drop. In this case, although the honest server drops one message, the attacker's observations will be exactly the same as if the devices could coordinate and only one device submitted setup messages. The second case is that d' submits a circuit-setup message to a different dead drop than d. A device-partitioning attacker can cause this, *e.g.,* if it gives one device an old address book where a now-occupied slot was free. In this case, one of the circuits' dead drops receives one less circuit, *i.e.,* becomes a "single"-access dead drop, which is covered by Rubato's noise. The attacker could obtain the same view by dropping a circuit-setup message on the network (even if there are no multiple devices involved).

Finally, there is a chance that there is no honest server on the route of the circuits that d and d' use. In this case, the attacker can anyway trace Alice's messages through the malicious mix servers, all the way to the dead drop; as a result, the attacker learns nothing new from partitioning Alice's devices. Rubato mitigates this risk by using sufficiently long mixnet routes (§4.5.5.3). □

4.5.5.2 Rogue Service Providers

In this section, we show that for any view of Rubato an attacker can observe by running service providers, the attacker can also gain the same view by dropping the messages users send on the network. The next section shows that Rubato's mixnet provides privacy in the face of such attacks. As Rubato does not rely on the providers to shuffle messages, a user's privacy is independent of the number of users connected to their provider. Therefore, it is safe to be the sole user of a bad provider.

Property 4.5.2. Consider an attacker that operates the user's service provider and their observations about the user's communication. A more restricted attacker, which controls only the network and the same set of mix servers (but not the provider), can obtain the same observations.

Proof. Let us consider all the ways a user's devices interact with their service provider. We show that, at all steps in the protocol, the attacker cannot learn any information that he cannot also learn by dropping the client's network messages.

First, clients retrieve the epoch number from their provider so that they can prime circuit-setup messages (§4.5.4.1). A malicious provider can lie about the epoch number or submit the circuit-setup messages at the wrong epoch. The client writes the epoch number at every onion layer of the setup message, and honest mix servers discard onions for the wrong epoch number. As a result, this provider's attack is equivalent to the network attacker simply dropping the user's circuit-setup messages (if an honest server exists). If there is no honest server on the route, even the restricted attacker can learn everything about the circuit simply by observing the setup message route from one malicious mix server to the next.

Second, all of a user's devices synchronize address books through the service provider. Clients update the user's address book before replacing circuits (§4.5.4.3), which happens independently of the user's buddies. On each update, the client uploads their address book under fresh encryption, padded to a fixed length (`MaxBuddies`), hence the service provider cannot tell whether it has changed. The provider can give stale address books to clients or otherwise break the transactional semantics made for synchronizing between clients; these attacks are variants of partitioning a user's devices (see Theorem 4.5.1).

Lastly, the client uses the provider to send and receive messages. When sending a message, the client submits one onion-encrypted message that does not contain any information about the intended circuit (the provider then broadcasts it on all circuits, §4.5.4.4). The service provider also serves the messages to the recipient's client by routing them through the mixchain that the client chooses when calling `CheckForMessages` (§4.5.4.5). The client's choice of the servers in this mixchain is independent of the user's buddy-relationships, hence it leaks nothing about them. As each mix server chooses a fresh nonce when shuffling messages, each mix server uses a different ephemeral key for every fetch; hence, the messages output from the fetch mixchain appear random to the service provider (even if the provider replays old requests). Furthermore, the client chooses a new key each time it calls `CheckForMessages` so that it always fetches messages from random-looking locations (even if the provider replays old headers). Therefore, the client's pattern of retrieving messages from the service provider appears random each time (unless the attacker also controls all of the mixchain servers, in which case it can anyway learn which messages the client fetches without controlling the user's provider). □

4.5.5.3 Differential Privacy

All messages that Rubato handles, except circuit setup, route through existing circuits. For these messages, the attacker observes the same communication pattern as in the circuit-setup round. Even if the attacker attempts to change the round's communication pattern by dropping a message, the first honest downstream server fills in an indistinguishable message in its place (§4.5.4.4). Therefore, as long as such a server exists, no new information leaks to the attacker during an epoch (after circuit establishment).

The remainder of this analysis is focused on the circuit-setup round. We split the analysis into two parts, first reasoning about passive attacks and then about active attacks. Our analysis resembles Karaoke's analysis [141], as Rubato's circuit-setup round resembles Karaoke's messaging rounds.

Passive Attacks. A passive attacker only observes network traffic and knows the secrets of the servers it controls. This attacker sees the number of circuits on every inter-server link and the number of circuits arriving at each dead drop hosted on the mix servers it controls. Intuitively, the passive attacker observes the same dead-drop access pattern, regardless of Alice's relationship with Bob: Alice and Bob choose message routes by using independent multi-device keys, and their circuits reach a dead drop that is accessed exactly twice (regardless of whether they are buddies). As long as the two circuits that Alice and Bob create route through sufficiently many honest servers, the mixnet unlinks these messages from Alice and Bob's identities. Property 4.5.3 formalizes this intuition.

Property 4.5.3. The passive attacker has the same probability of observing event $\mathcal{O}$, regardless of whether Alice and Bob are buddies, except for a small error probability, δ_p that diminishes exponentially with the route length.

Proof. Rubato clients uniformly choose a server for each hop. Consider a route length l such that each circuit-setup message routes through at least two honest servers with probability at least $1 - \frac{\delta_p}{4}$. Karaoke's analysis shows that in this case, the passive attacker has an equal chance to observe the message volumes on network links regardless of whether Alice and Bob are buddies [141, §5.2]. Since there are four circuit-setup messages, potentially coordinated between Alice and Bob (two from Alice and two from Bob), then by the union bound, the chance the premise for this analysis does not hold is less than $4\frac{\delta_p}{4} = \delta_p$. Since each mix server is malicious with probability at most f (§4.5.3.1), we need $1 - \frac{\delta_p}{4} \geq 1 - f^l - l(1-f)f^{l-1}$. Thus, $\delta_p \leq 4f^l + 4l(1-f)f^{l-1}$. □

As in Karaoke [141], to resist passive attacks, Rubato requires two honest mix servers on the route of the circuit connecting Alice to the dead drop where she exchanges messages with Bob. The error probability δ_p captures the chance that there are no such honest servers.

Active Attacks. Consider an active attacker that disrupts circuit setup by dropping messages. In case the attacker controls the servers hosting the dead drops that Alice and Bob use, then dropping some of their circuit-setup messages shows different dead-drop access patterns (depending on whether Alice and Bob are buddies). For example, if Alice and Bob were buddies, then dropping Alice's messages would result in two dead drops being accessed by only one circuit (from Bob). But if they were not buddies, then there will be just one dead drop accessed by both of Bob's circuits. The noise that honest servers inject obscures these observations (the attacker cannot drop the noise due to the Bloom filter checks, §4.5.4.2).

Property 4.5.4. Rubato is differentially private. There are ϵ and δ_a such that the inequalities from Equation 4.4 hold.

Proof. Karaoke's analysis shows the differential privacy guarantees for a mixnet with Rubato's topology [141, Thm.8]. We apply it here in the context of circuit-setup messages: for every pair of circuits, whose dead drops Alice might change depending on her communication with Bob, the attacker's observations are (ϵ, δ_a)-differentially private. Since Alice's client sets up multiple circuits, and might use any of them to communicate with Bob, we need to generalize Karaoke's analysis. Consider the two scenarios that Rubato obfuscates:

If Alice is buddies with Bob and wants to claim they are not buddies, then only the two dead drops of the circuits coordinated with Bob would change, so [141, Thm.8] applies directly:

$$\Pr[\mathcal{O}|\mathrm{A} \leftrightarrow \mathrm{B}] \leq e^{\epsilon} \Pr[\mathcal{O}|\mathrm{A} \not\leftrightarrow \mathrm{B}] + \delta_a$$

Else, Alice and Bob are not buddies, but she should be able to claim they are buddies. Alice's client sets up `MaxBuddies` pairs of circuits (Figure 4.5.4) and could use any pair to communicate with Bob. Let p_i be the probability that Alice's client uses pair i to communicate with Bob:

$$\Pr[\mathcal{O}|\mathrm{A} \not\leftrightarrow \mathrm{B}] = \sum_i \Pr[\mathcal{O}|\mathrm{A} \not\leftrightarrow \mathrm{B} \wedge i]\, p_i$$

Applying [141, Thm.8] on the two circuits in pair i for each element in the sum, we get:

$$\leq \sum_i (e^{\epsilon} \Pr[\mathcal{O}|\mathrm{A} \leftrightarrow \mathrm{B} \wedge i] + \delta_a)\, p_i = e^{\epsilon} \Pr[\mathcal{O}|\mathrm{A} \leftrightarrow \mathrm{B}] + \delta_a$$

□

4.5.6 Implementation

We implemented a prototype of Rubato in Go, on top of Yodel's mixnet framework [142], in 20k lines of code. The prototype uses ChaCha20 for symmetric onion encryption, the NaCl box primitive to generate circuit setup onions, and the Blake2b hash function. Communication between clients to service providers and between Rubato's mixnet servers uses gRPC with TLS 1.3 for transport security. The service provider uses BadgerDB to implement transactions and manage user state.

Our implementation includes two types of clients, one for desktops and one for mobile devices. The desktop client is a command-line program, and the mobile client is built for Android by using the `gomobile` tool.

Memory Usage. One prominent challenge in implementing Rubato has been minimizing its memory footprint. With 3M users and 100 mix servers, each server needs to keep track of at least 42 million pieces of cryptographic state per epoch:

$$100 \text{ circuits} \times 3\,000\,000 \text{ users} \times \frac{1}{100 \text{ servers}} \times 14 \text{ hops} = 42\text{M}.$$

Initially, we used AES-GCM to implement Rubato's circuits, but its state is 512 B, thus resulting in at least 20 GB of memory usage per mixnet server. To reduce memory, we replaced AES-GCM with ChaCha20, which requires only 32 B per state, reducing this memory usage to 1.3 GB per server but increasing CPU usage, due to lack of hardware acceleration.

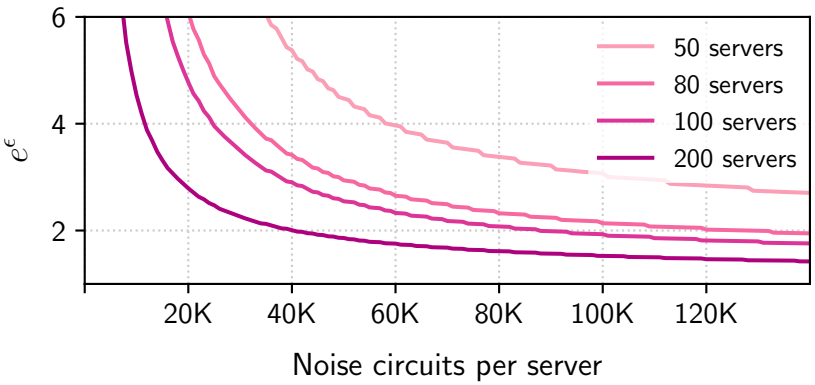


Figure 4.5.7 – Privacy vs. noise for different deployment sizes after 245 epochs with active attacks and 9,600 epochs of passive observations. We assume $f = 20\%$ malicious servers, and fix $\delta = 10^{-4}$ and the route length to be 14 hops.

4.5.7 Parameter Selection

We set Rubato to resist $f = 20\%$ malicious servers and the mixnet path length to be 14 hops (similarly to previous works with the same mixnet topology [141, 142], that require two honest servers in a route). We also set the fetch mixchain length (§4.5.4.5) to be 11, which provides a smaller chance of error of the routes via the mixnet (the fetch mixchain requires only one honest server hence is shorter). A conversation between two users requires a pair of circuits, each carrying a payload of 64 bytes each round. Out of the 128-byte payload, 2 bytes are reserved by the pseudorandom header indicating whether the message is real or cover traffic (see §4.5.4.5), 12 bytes are reserved for the end to end authentication code, and 12 bytes are reserved for a nonce. As a result, recipients receive a 102-byte encrypted text message per conversation.

Noise in Practice. To decide the actual noise volume that Rubato mix servers need to induce, we set a goal for differential privacy δ and fix the number of epochs the system would support. Given the number of layers in Rubato's deployment, we compute the passive analysis's δ_p (Theorem 4.5.3). We set the remainder, $\delta_a = \delta - \delta_p$, to be the target error probability for differential privacy under active attacks (Theorem 4.5.4). We then compute the mean amount of noise in the system following [141, Thm.8], and we compose over the number of active attacks on a user that the system needs to sustain (by using differential privacy's composition theorem [81, 3.20]). As dishonest servers do not contribute to the noise, we have each server generate $\frac{1}{1-f}\times$ noise of their relative portion (so we obtain sufficient noise from the honest servers).

Clients detect active attacks because they never receive an acknowledgment when the attacker drops their circuit-setup message (the hash of the dead-drop address, revealed at the last mix server §4.5.4.2). This enable clients to alert users who can take measures to counter the user-privacy risk (similar to those proposed by Lazar et al. [141], such as ceasing communication or switching service providers). Therefore, we target resisting many more passive attacks than active attacks.

Figure 4.5.7 illustrates Rubato's differential privacy guarantee (ϵ, δ) as a function of the number of noise circuits that each server creates. It shows the composition over 245 epochs of active attacks and 9 600 epochs of passive observations. These parameters provide a comparable privacy analysis to Karaoke [141], as it is configured to resist the same number of active attacks and to sustain passive attacks for about a year (we assume Rubato's epochs are at least 1-hour

long). With 100 servers, each honest mix server needs to create 88 000 noise circuits in every epoch to provide ($\epsilon = \ln 2, \delta = 10^{-4}$)-differential privacy (the same ϵ, δ as Vuvuzela and an improvement over Karaoke's $\epsilon = \ln 4, \delta = 10^{-4}$ [141, 220]).

4.5.8 Evaluation

We use the prototype to evaluate Rubato's performance and costs. Our experiments answer the following questions:

1. What are the throughput and latency Rubato can achieve?
2. How does Rubato scale with the number of servers?
3. What are the system's deployment costs?
4. What are the costs for a mobile client in terms of battery consumption and network usage?
5. What is the cost for a client catching up to old messages after having been offline?

Our evaluation has two parts: first, we focus on the system's back-end, *i.e.,* the service providers and mixnet, and then on the mobile client.

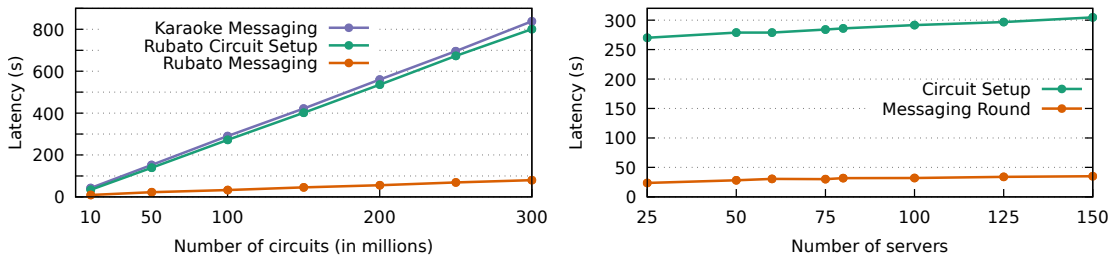
4.5.8.1 Back-End Performance and Costs

Setup. We deploy mixnet servers evenly split across 3 EC2 regions across the U.S. and one in Europe: `us-east-1`, `us-east-2`, `us-west-2`, `eu-west-1`. Each server is an `r5.8xlarge` VM with an Intel Platinum 8000 3.1 Ghz CPU with 32 cores, 256 GB of memory, and a 10 Gbit/s network link. We test Rubato with 25-150 mixnet servers. We evaluate with a single service provider on `us-east-1`: As service providers only buffer and relay messages, and do not participate in processing messages through the mixnet, the performance of Rubato's back-end does not depend on the number of service providers. Only clients connected through an overloaded provider will experience performance issues.

We simulate hundreds of millions of circuits by having mixes create extra circuits during circuit setups. Assuming each user has up to 50 buddies, and hence requires 100 circuits, this corresponds to relationships between millions of users. Although clients do not use these circuits, they correspond to a real conversation load on the back-end. We set Rubato's system parameters as described in §4.5.6.

Throughput and latency. We measure back-end latency for a given circuit load in deployment of 100 mixnet servers. To measure the latency on the back-end, we measure the time since the service provider submits a message until the mixnet round completes and returns the result to the provider, plus the time needed for the fetch protocol (§4.5.4.5). Users will experience additional latency depending on their schedules and the network round-trip times to their service providers.

In Figure 4.5.8a, we observe that the Rubato's messaging round latency is 32.4s, 55.9s and 79.83s for 100M, 200M and 300M circuits, respectively. The measured latencies have two components: The first, larger, component mixes the users' messages and routes them from the source to the destination service provider; this represents the majority of the duration of Rubato's messaging round (28.7s, 50.8s and 71.3s for 100M, 200M and 300M circuits). The second, smaller, component ($\approx$ 11% of the total duration) is the time spent on running Rubato's fetch protocol (§4.5.4.5). For this second part, we model an average load of clients running the fetch protocol: We simulate fetches at the end of each round, and we wait for all messages of a



(a) Latency of the circuit setup round and of messaging rounds with respect to the number of users.

(b) Scalability of the back-end. The server load is constant (1 million circuits / server).

Figure 4.5.8 – Evaluation of the back-end

round to be fetched before moving on to the next round. Finally, assuming the average user sets up 100 circuits (to communicate with up to 50 buddies), Rubato could support 1M-3M users with these latencies.

We compare this latency to Yodel [142] that also relies on symmetrically-encrypted onions to deliver messages through a mixnet. However, in Yodel, the receiver gets messages directly from the dead drop (at the cost of requiring synchronicity between clients). Rubato and Yodel also differ in the number of circuits a user would need and the message size: In Rubato, users can operate asynchronous conversations with many buddies in parallel, and the system delivers 102 B text messages to a buddy (on two circuits) at every round. In contrast, Yodel targets voice calls; it allows just one conversation at a time and uses 64 byte messages from the voice encoder. Yodel's latency for 2M users is 1.6s, whereas Rubato's latency is 55.9 seconds, a 35× increase. We attribute Rubato's performance gap from Yodel to the 50× increase in the number of circuits and to the additional latency from delivering messages from the dead drop to the recipient's service provider via the mixnet.

The duration for setting up the circuits in Rubato is 13.3 minutes for 300 million circuits (Figure 4.5.8a). As circuit setup is relatively infrequent (*e.g.*, once a day), it can run in the background and stretch up to an epoch's duration. Rubato's circuit setup is similar to a messaging round in Karaoke [141], though there is a difference in the payload size (Rubato's payload is about 200 bytes smaller than Karaoke). For the same setup, where each of 3M users has 50 buddies, a communication round in Karaoke takes 14 minutes (Figure 4.5.8a).

Scalability. We test how Rubato scales with the number of users by measuring the latency for varying deployment sizes of 25-150 mixnet servers and by keeping the number of circuits per server constant at 1M (so the load on the system increases proportionally to the number of servers). Figure 4.5.8b shows that Rubato scales well: it can support additional users at almost the same latency by proportionally increasing the number of servers. We attribute the slight latency increase to the fact that shuffling messages together requires each server, in every hop along the mixnet route, to wait for inputs from all other servers that processed messages at the previous hop.

Deployment costs. With 300 million circuits, each of the 100 mixnet servers sends at about 2 Gbps at peak usage, for a network usage of 13.4 GB per messaging round. In this deployment, setting up circuits for one epoch uses 47.5 GB of network data per mixnet server.

Service providers buffer messages for the users connected to them. For a user who generates circuit-setup messages for the next 30 epochs (*i.e.*, the next month if epochs last a day), the

storage requirement for circuit setup is 2.1 MB. Furthermore, if messaging rounds occur every minute, then storing a month's worth of messages amounts to 264 MB per user.

4.5.8.2 Mobile clients

We turn to evaluate running Rubato's mobile client. We focus on two metrics: the effect on battery life and on network usage. We evaluate the mobile client on a Pixel 4 mobile device that runs the stock Android 10 operating system. Our tests include cellular (3G with HSDPA) and Wi-Fi networks.

Battery usage. We explore the effect of different schedules on battery life. To evaluate battery consumption, we connected the mobile device to a USB power meter (UM25C) and, roughly every 1 second, we collected energy consumption.

We force enabled the *doze mode* to reduce noise from apps running in the background; this is the battery-saving optimization that typically runs when the device's USB port is unplugged. Running Rubato's app in this mode means that the client's transmissions can change without adhering to the schedule. Therefore, we excluded our app from doze mode (Android's `AlarmManager` provides APIs to avoid it).

First, we measure the baseline energy consumption when the phone is fully charged and idle, with the screen turned off, and without Rubato's client. We observe that, after one hour, the idle phone consumes about 310 mWh of energy, both when the phone uses Wi-Fi and cellular networks. Then, we run Rubato's client with different schedules. At the beginning of the experiment, the client uploads circuit-setup messages for the next epoch. Then, each time the schedule triggers, the client sends a message and downloads pending messages (following Figure 4.5.3).

Figure 4.5.9 shows the energy consumption over an hour for different schedules compared to the baseline. The results for the 1-minute and 5-minute schedules were collected for a 1-minute mixnet round interval. To accommodate the 30-second schedules, we run these measurements with a 30-second mixnet round interval. It is apparent from the figure that as the schedule becomes more frequent, the energy consumption increases, especially when using cellular networks. A 1-minute schedule on the cellular network uses an extra 60 mWh compared to the baseline; this is a 19% increase. For comparison, the Pixel 4 battery capacity is 14 000 mWh.

We observe that energy cost from running Rubato is substantially reduced when the client runs on a five-minute schedule. We hypothesize that this enables the phone to hibernate and save power. The energy consumption is more modest when using Wi-Fi, which is more energy-efficient [127].

Network usage.

To quantify the client's network usage, we monitored the link between the client and the service provider. Every time the schedule trigger, the client (1) uploads one message (split into two) to the service provider and (2) downloads two messages per round (corresponding to 1 buddy, see Figure 4.5.5). Consequently, different schedules affect the message upload volume, which ranges from 25 KB/h with a 5 minute send schedule to 130 KB/h with a 1 minute schedule (and 1 minute mixnet round time). The download volume is about 140 KB/h on the two schedules (as our client fetches 2 messages per round). The client also sends about 110 KB for the circuit setup per epoch. For a month's worth of primed circuits, this phase costs 3.3 MB.

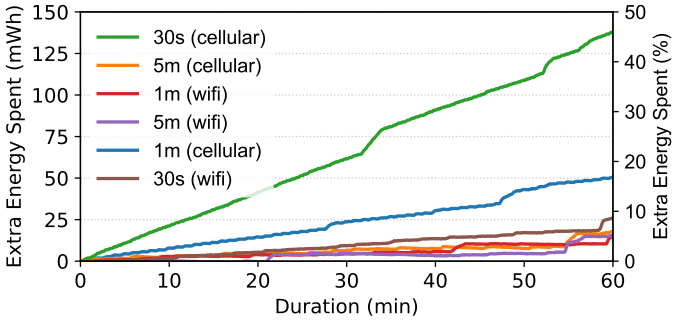


Figure 4.5.9 – Energy consumption of Rubato's client on a Pixel 4 phone for different schedules and network types. The graph shows how much more energy is spent running a schedule compared to leaving the phone idle.

We estimate that clients use a total of 220 MB to 290 MB of bandwidth per month, depending on the schedule. This is over two orders of magnitude less than Vuvuzela's and Alpenhorn's dialing protocols, which require about 7 MB per round or 300 GB per month [143, 220] for a primitive form of asynchronous messaging on the same schedule. We believe that Rubato's network usage is moderate and compatible with mobile data packages.

4.5.9 Limitations

For the sake of performance, Rubato provides a differential-privacy guarantee, which is weaker than that of some related works [138]. In Rubato, when users communicate, they might reveal small, quantified statistical information to an adversary. Over time, users can deplete a privacy budget. Once it reaches zero, users must stop using the system the system or risk revealing to the adversary whether they are in an active communication or not. Although we designed the system to work for a long time (§4.5.5), this approach can be unsatisfactory.

Furthermore, in Rubato, users must communicate following a personal "safe schedule" (§4.5.2.1). Safe schedules, however, can still be unpractical; the safe schedule that we provide as an example is a fixed schedule. Even when we hint that other safe schedules are possible (§4.5.2.1), further study is needed to propose practical safe schedules.

4.5.10 Conclusion

In this subsection, we have presented Rubato, a metadata-private communication system based on a mixnet. This system provides a differential privacy guarantee and can scale to support a large user base. The major advance in Rubato is that it supports mobile devices that can go offline, have power and network constraints, and that complement other devices a user has. Rubato achieves this by introducing the primed circuits and via circuit tagging techniques that enable bridging asynchronous clients with a synchronous mixnet through untrusted service providers. Much like standard messaging applications, Rubato enables users to run clients on multiple devices. We implement a prototype of Rubato and show that it can support a large user-base, with latency on the order of a minute. Experiments with a Pixel 4 smartphone demonstrate that Rubato can accommodate mobile clients' network and power constraints.

4.6 Conclusion

In this chapter, we have proposed two novel systems: PriFi and Rubato.

PriFi is an ACN tailored for organizational networks: it protects users from eavesdropping and tracking attacks. PriFi exploits the characteristics of (W)LANs to provide low-latency, traffic-agnostic communication against a strong adversary. Unlike most other ACNs, the user traffic remains on its usual path; the added latency is due to software processing and not due to additional network links. We have also presented novel low-latency techniques for protecting against malicious insider attacks and equivocation attacks. Due to the anonymization technique used, PriFi has a high network cost and does not scale to more than a few hundred users.

Rubato is an ACN for instant messaging that can scale to millions of users. Unlike the related work, it supports multiple uncoordinated and asynchronous clients per user. By respecting their power and network constraints, this enables mobile devices to participate in the system. Rubato achieves this by introducing the primed circuits and circuit tagging techniques, which enable bridging asynchronous clients with a synchronous mixnet through untrusted service providers. Due to the use of dummy circuits to hide real conversations, the system has a high bandwidth usage on the back-end.

5. Conclusion

In this thesis, we have explored problems on the topic of communication metadata. We have focused on two complementary approaches: raising awareness by demonstrating attacks, and presenting novel solutions.

In Chapter 2, we have presented an inference attack on the communications of Bluetooth wearable devices. We demonstrated that a passive adversary, for instance a nosy neighbor or a local advertiser, can infer personal and health-related data from encrypted communications. This adversary is able to recognize devices and hence track users, to identify application openings on smartwatches, and even to recognize fine-grained actions within applications. More fundamentally, we have shown how all the wearable devices that we analyzed were insufficiently protected against the threat of traffic analysis. This confirms that the analysis of communication metadata can pose a threat to user privacy, even in domains where traffic analysis was not previously considered an immediate threat.

In Chapter 3, we have presented a ciphertext format (PURBs) and a padding function (Padmé) for reducing metadata leakage. In particular, we have demonstrated that, with appropriate trial-decryption steps and specific header structure, a ciphertext format can be made indistinguishable from random bits hence not leak any metadata, except for its length. Then, we have proposed a padding function for reducing the information leakage through the length. We have shown how it reduces the number of bits leaked, compared to traditional padding functions, and that it maintains a reasonable cost in practice.

In Chapter 4, we built communication systems that protect some communication metadata. More precisely, we developed Anonymous Communication Networks (ACNs) to hide the communicating entities.

First, we have focused on the setting of organizational networks: LANs and WLANs in non-governmental organizations, companies, and universities. We realized that current ACNs are poorly suited to protect communication metadata in this context: Most rely on an anytrust chain of server that must be spread across different jurisdictions in order to maximize collective trustworthiness. This topology adds latency to communications, limits the applications that can communicate over the network, and adds friction for users. We developed PriFi, an ACN for the setting of local-area networks. PriFi uses a new DC-net topology to protects the identity of users. It enables packets from the users to remain on their usual network path and conserves the desirable property of having servers geographically spread in different jurisdictions. As a result, PriFi achieves low latency and can support applications such as VoIP.

Finally, we have focused on the problem of metadata in instant mobile messaging (similar to WhatsApp). The state of the art in scalable anonymous communication require users to follow a fixed global schedule to defend against intersection attacks. This requirement might be acceptable for servers that are always online and available, but it excludes mobile users:

first, because such devices cannot be expected to be always online; second, because a unique global schedule cannot accommodate the diverse constraints (*e.g.,* in terms of battery life) that mobile users have. We developed Rubato: an ACN for "instant" messaging that can scale to millions of users, has a moderate latency, and that enables users to follow their own schedule. Rubato provides a unique anonymity set and resists intersection attacks, even if users disconnect. This work enables mobile devices to participate in large-scale Anonymous Communication Networks, hence enables protecting the communication metadata of a new important class of users.

Limitations

Technical Limitations

In Chapter 2, we have presented a traffic-analysis attack that enables an adversary to infer sensitive information from the encrypted communications of Bluetooth wearable devices. This experimental work has several limitations, listed in Section 2.8.2. We highlight two important points: First, due to practical constraints, our analysis is limited to 13 devices and 80 applications. Although we performed a preliminary experiment to validate that our results generalize (§2.6.2.2), we cannot definitely answer on the generalizability of our results. We recall that it was not our primary goal; we sought to demonstrate a privacy problem with a common set of wearable devices. When choosing the set of devices used in our experiments, we made a best-effort approach to cover different vendors and device types, as discussed in §2.4. We acknowledge, however, that the success rate and practicality of the attack depends on the choice and number of devices. The second limitation stems from the powerful adversary used. This adversary is able to eavesdrop on Bluetooth communications, with a high capture rate. By using hardware available today, this implicitly assumes a near-by eavesdropper with expensive equipment. However, there is a trend towards less expensive Bluetooth sniffers (§2.8.1). The choice of adversary also has an effect on the defenses (§2.7): their success rates should be taken as an upper-bound, given our set of features. In the same vein, if a weaker adversary were to be considered, the simple defenses we propose might provide sufficient protection.

In Chapter 3, we have presented a ciphertext format and a padding function to reduce metadata leakage. Our design is a generic approach that is likely to be outperformed by solutions tailored for their application domain. In particular, Padmé is a generic approach that, according to our observation, provides a good trade-off in some domains (§3.5), while having better theoretical properties than trivial solutions. Our goal with PURBs was to demonstrate the feasibility of a ciphertext format that leaks "almost nothing". Due to time constraints, the evaluation of the uses-cases of Padmé and PURBs is not extensive.

PriFi (§4.4) has high bandwidth costs and requires clients to be synchronous. With the requirement of low latency and due to the protocol design, our implementation does not scale to more than a few hundred participants (§4.4.10). Malicious clients who blatantly cheat on the protocol can be detected at a cost (stopping all communications and running an additional protocol, §4.4.6). Hence, PriFi is best suited to be a close-membership system. A slow client can degrade the overall network throughput, but it might be difficult to determine if such client is maliciously degrading the network. Therefore, PriFi is best suited for small-scale high-performance communications in settings with spare bandwidth. An example of such setting is a governmental or non-governmental organization with a local-area network on which devices are authenticated, but loosely trusted.

Rubato (§4.5) provides a differential-privacy guarantee (§4.5.3.2). Over time, users can deplete a privacy budget. Once it reaches zero, users must stop using the system altogether to avoid revealing statistically significant observations to the adversary. Although we designed the system to work for a long time (§4.5.6), this approach can be unsatisfactory. Furthermore, in Rubato, users must communicate following a "safe schedule" (§4.5.2.1). Unlike in related works, this schedule is not global and can be chosen by each user. Safe schedules, however, can still be impractical; the safe schedule that we provide as an example is a fixed schedule. Even when we hint that other safe schedules are possible (§4.5.2.1), further study is needed to propose practical safe schedules.

Societal Limitations

Law enforcement agencies often use lawful interception techniques on communications as part of their mandate to fight crime. Communication systems that protect contents and metadata from third parties could hamper the legitimate operations of these agencies.

The purpose of the systems we propose is not to evade lawful surveillance: It is to protect user privacy from unauthorized third parties (*e.g.*, hackers, advertisers) by removing sensitive metadata from communications. We deem the presence of such metadata to be incompatible with the respect of user privacy, which is a fundamental right in many democratic societies [59]. If the lawful surveillance operations rely on the same sensitive metadata, we believe that such operations must evolve.

For instance, we believe that an open, auditable process could be created to enable law enforcement to lawfully access specific data. An example of such system in the context of mobile phone data collected at cell towers is presented by Segal *et al.* [190, 191]; the requirements of such a process are further discussed by Feigenbaum *et al.* [87]. Yet, at the time of writing this thesis, we believe that some prerequisites are missing in today's societies: For instance, if a court should be able to order a lawful surveillance and have the technical means to decrypt some encrypted data, an auditable infrastructure to manage the court's cryptographic identities must exist.

Currently, there is no practical system that protects all communication metadata while supporting low-latency traffic at a large scale. Thus, we focus our research towards this technical problem. We acknowledge that the interaction with law enforcement is left to be defined.

Impact outside of research

We have conducted responsible disclosure regarding the traffic-analysis attack on wearable devices (§2). We contacted all relevant device manufacturers and app developers with our findings. This generated interest, notably around potential solutions to the problem. One manufacturer awarded us a bug bounty. This supports the hypothesis that our objective to raise awareness has been achieved. We have also made available the datasets and attacks in order to facilitate the development of defenses [24].

Regarding PURBs and Padmé (§3), the developers of Sequoia PGP, a modern OpenPGP library coded in Rust, integrated Padmé as one option to pad their ciphertexts [193].

Part of the development of PriFi was done in collaboration with the ICRC, in particular the fine-tuning and benchmarking of performance with data from real ICRC sites. A demo of the working prototype was presented at the ICRC Headquarters in Geneva on the 10th of

September 2018. Two internal demonstrations took place during two EPFL events (EPFL Summer Research Institute 2017 and EPFL IC Research Days 2018). The latter demo was awarded a prize. The research around PriFi also led to a patent [89].

PURBs (§3), PriFi (§4.4) and Rubato (§4.5) are systems that can be reused; we open-source them to facilitate further software development and research [21, 22, 162]. PURBs and Rubato are research prototypes, whereas PriFi has been pushed to be a slightly more mature product [21].

Reproducibility

In order to ensure full reproducibility, we have made available all datasets and tools used in this thesis[1]: [24] for Chapter 2, [162] for Chapter 3, and [21, 22] for Chapter 4.

Funding

EveryByteMatters (Chapter 2) was supported in part by grant 172541 of the Swiss National Science Foundation (SNF).

PURBs (Chapter 3) was supported in part by grant 2017-201 of the Strategic Focal Area "Personalized Health and Related Technologies (PHRT)" of the ETH Domain and by grants from the AXA Research Fund, Handshake, and the Swiss Data Science Center.

PriFi (§4.4) was supported in part by U.S. National Science Foundation grants CNS-1407454 and CNS-1409599, U.S. Department of Homeland Security grant FA8750-16-2-0034, U.S. Office of Naval Research grants N00014-18-1-2743 and N00014-19-1-2361, and by the AXA Research Fund.

Closing Words

The problem of communication metadata is pervasive and difficult to solve with a holistic approach. Hiding communication metadata is fundamentally costly. When developing practical systems, these high costs raise the question about the necessity of protecting metadata. In particular, should all metadata be hidden, or only some of them (*e.g.,* the traffic patterns, or the communicating entities) ? What is an ambitious, privacy-preserving but practical goal for the next mainstream communication system willing to protect communication metadata, *e.g.,* the next Signal?

These remain open questions. We hope that the contributions in this thesis promote and help build systems that carefully consider the privacy threats stemming from their communication metadata.

[1] With the exception of one private dataset used in the evaluation of PriFi.

Bibliography

[1] M. Abdalla, M. Bellare, and P. Rogaway. The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. In *Cryptographers' Track at the RSA Conference*, 2001.

[2] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac. Peek-a-Boo: I See Your Smart Home Activities, Even Encrypted! In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2020.

[3] H. Aksu, A. S. Uluagac, and E. Bentley. Identification of Wearable Devices with Bluetooth. In *IEEE Transactions on Sustainable Computing*, 2018.

[4] W. Albazrqaoe, J. Huang, and G. Xing. Practical Bluetooth Traffic Sniffing: Systems and Privacy Implications. In *ACM Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2016.

[5] W. Albazrqaoe, J. Huang, and G. Xing. A Practical Bluetooth Traffic Sniffing System: Design, Implementation, and Countermeasure. In *IEEE/ACM Transactions on Networking*, 2018.

[6] A. Alshehri, J. Granley, and C. Yue. Attacking and Protecting Tunneled Traffic of Smart Home Devices. In *ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2020.

[7] Anacast. ProxBook: Proximity Marketing in Airports and Transportation. https://unacast.s3.amazonaws.com/Proxbook_Report_Q3_2016.pdf, 2016. Accessed: 2020-09-20.

[8] S. Angel, H. Chen, K. Laine, and S. Setty. PIR with Compressed Queries and Amortized Query Processing. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[9] S. Angel, D. Lazar, and I. Tzialla. What's a little leakage between friends? In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2018.

[10] S. Angel and S. Setty. Unobservable Communication Over Fully Untrusted Infrastructure. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2016.

[11] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen. Nearby Threats: Reversing, Analyzing, and Attacking Google's 'Nearby Connections' on Android. In *Network and Distributed Systems Symposium (NDSS)*, 2019.

[12] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen. Bias: Bluetooth Impersonation Attacks. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[13] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen. Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy. In *ACM Transactions on Privacy and Security (TOPS)*, 2020.

[14] D. Antonioli, N. O. Tippenhauer, and K. B. Rasmussen. The KNOB is Broken: Exploiting Low Entropy In The Encryption Key Negotiation Of Bluetooth BR/EDR. In *USENIX Security Symposium*, 2019.

[15] Apple. iMessage and Facetime and Privacy. https://support.apple.com/en-us/HT209110, 2021. Accessed: 2021-02-12.

[16] N. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster. Keeping The Smart Home Private with Smart(er) IoT Traffic Shaping. In *Privacy Enhancing Technologies Symposium (PETS)*, 2019.

[17] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster. Spying on The Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic. In *arXiv preprint 1708.05044*, 2017.

[18] D. F. Aranha, P.-A. Fouque, C. Qian, M. Tibouchi, and J.-C. Zapalowicz. Binary Elligator Squared. In *International Workshop on Selected Areas in Cryptography (SAC)*, 2014.

[19] A. Bahramali, R. Soltani, A. Houmansadr, D. Goeckel, and D. Towsley. Practical Traffic Analysis Attacks on Secure Messaging Applications. In *arXiv preprint 2005.00508*, 2020.

[20] L. Barman, I. Dacosta, M. Zamani, E. Zhai, A. Pyrgelis, B. Ford, J. Feigenbaum, and J.-P. Hubaux. PriFi: Low-latency Anonymity for Organizational Networks. In *Privacy Enhancing Technologies Symposium (PETS)*, 2020.

[21] L. Barman, I. Dacosta, M. Zamani, E. Zhai, A. Pyrgelis, B. Ford, J. Feigenbaum, and J.-P. Hubaux. Repository for "PriFi". https://github.com/dedis/prifi, 2020. Accessed: 2021-05-01.

[22] L. Barman, I. Dacosta, M. Zamani, E. Zhai, A. Pyrgelis, B. Ford, J. Feigenbaum, and J.-P. Hubaux. Repository for "PriFi Experimental Results". https://github.com/lbarman/prifi-experiments, 2020. Accessed: 2021-05-01.

[23] L. Barman, A. Dumur, A. Pyrgelis, and J.-P. Hubaux. Every Byte Matters: Traffic Analysis of Bluetooth Wearable Devices. In *International Conference on Ubiquitous Computing (UbiComp)*, 2021.

[24] L. Barman, A. Dumur, A. Pyrgelis, and J.-P. Hubaux. Repository for "Every Byte Matters: Traffic Analysis of Bluetooth Wearable Devices". https://wearable.lbarman.ch, 2021. Accessed: 2021-05-01.

[25] L. Barman, M. Kol, D. Lazar, Y. Gilad, and N. Zeldovich. Rubato: Metadata-Private Messaging for Mobile Devices. In *Under submission*, 2021.

[26] L. Barman, M. Zamani, I. Dacosta, J. Feigenbaum, B. Ford, J.-P. Hubaux, and D. Wolinsky. PriFi: A Low-latency and Tracking-Resistant Protocol for Local-Area Anonymous Communication. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2016.

[27] A. Barth, D. Boneh, and B. Waters. Privacy In Encrypted Content Distribution Using Private Broadcast Encryption. In *International Conference on Financial Cryptography and Data Security*, 2006.

[28] J. K. Becker, D. Li, and D. Starobinski. Tracking Anonymized Bluetooth Devices. In *Privacy Enhancing Technologies Symposium (PETS)*, 2019.

[29] T. Be'ery and A. Shulman. A Perfect CRIME? Only Time Will Tell. https://owasp.org/www-pdf-archive/A_Perfect_CRIME_TIME_Will_Tell_-_Tal_Beery.pdf, 2013. Accessed: 2021-04-07.

[30] M. Bellare and C. Namprempre. Authenticated Encryption: Relations Among Notions and Analysis of The Generic Composition Paradigm. In *Journal of Cryptology*, 2008.

[31] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: Elliptic-Curve Points Indistinguishable from Uniform Random Strings. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2013.

[32] Bluetooth SIG. Bluetooth Core Specification v5.1. In *Bluetooth Special Interest Group (SIG)*, 2016.

[33] Bluetooth SIG. Bluetooth Range Calculator. https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/range/#estimator, 2020. Accessed: 2020-11-09.

[34] M. Blum, P. Feldman, and S. Micali. Non-Interactive Zero-Knowledge and its Applications. In *ACM Symposium on Theory of Computing (STOC)*, 1988.

[35] D. Boneh. The Decision Diffie-Hellman Problem. In *International Algorithmic Number Theory Symposium (ANTS)*, 1998.

[36] D. Boneh, C. Gentry, and B. Waters. Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys. In *Advances in Cryptology - Crypto*, 2005.

[37] R. Botsman. Big Data Meets Big Brother as China Moves to Rate its Citizens. https://www.wired.co.uk/article/chinese-government-social-credit-score-privacy-invasion, 2017. Accessed: 2021-02-12.

[38] J. Brooks. Ricochet: Anonymous Instant Messaging for Real Privacy. https://ricochet.im, 2016. Accessed: 2021-04-07.

[39] E. Brown. Most Businesses are Tracking Customers yet don't Tell them. https://www.zdnet.com/article/most-businesses-are-tracking-customers-yet-dont-tell-them/, 2020. Accessed: 2021-02-12.

[40] X. Cai, R. Nithyanand, and R. Johnson. CS-BuFlo: A Congestion Sensitive Website Fingerprinting Defense. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2014.

[41] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.

[42] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2012.

[43] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. RFC 4880, IETF, 2007.

[44] G. Celosia and M. Cunche. Saving Private Addresses: an Analysis of Privacy Issues In The Bluetooth-Low-Energy Advertising Mechanism. In *EAI International Conference on Mobile and Ubiquitous Systems (MobiQuitous)*, 2019.

[45] G. Celosia and M. Cunche. Discontinued Privacy: Personal Data Leaks In Apple Bluetooth-Low-Energy Continuity Protocols. In *Privacy Enhancing Technologies Symposium (PETS)*, 2020.

[46] Y.-C. Chang, K.-T. Chen, C.-C. Wu, and C.-L. Lei. Inferring Speech Activity from Encrypted Skype Traffic. In *IEEE Global Telecommunications Conference (GLOBECOM)*, 2008.

[47] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. In *Communications of the ACM (CACM)*, 1981.

[48] D. Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. In *Journal of Cryptology*, 1988.

[49] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig. HORNET: High-Speed Onion Routing at The Network Layer. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015.

[50] C. Chen, D. E. Asoni, A. Perrig, D. Barrera, G. Danezis, and C. Troncoso. TARANET: Traffic-Analysis Resistant Anonymity at The Network Layer. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018.

[51] W. Chen and R. A. Popa. Metal: A Metadata-Hiding File-Sharing System. In *Network and Distributed Systems Symposium (NDSS)*, 2020.

[52] G. Cherubin, J. Hayes, and M. Juarez. Website Fingerprinting Defenses at The Application Layer. In *Privacy Enhancing Technologies Symposium (PETS)*, 2017.

[53] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private Information Retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, 1995.

[54] J. Classen and M. Hollick. Inside Job: Diagnosing Bluetooth Lower Layers Using Off-the-Shelf Devices. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2019.

[55] J. Classen, D. Wegemer, P. Patras, T. Spink, and M. Hollick. Anatomy of a Vulnerable Fitness Tracking System: Dissecting The Fitbit Cloud, App, and Firmware. In *ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*, 2018.

[56] D. Cole. We Kill People Based on Metadata. https://www.nybooks.com/daily/2014/05/10/we-kill-people-based-metadata/, 2014. Accessed: 2017-12-12.

[57] M. Cominelli, F. Gringoli, P. Patras, M. Lind, and G. Noubir. Even Black Cats Cannot Stay Hidden In The Dark: Full-Band De-anonymization of Bluetooth Classic Devices. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[58] M. Cominelli, P. Patras, and F. Gringoli. One GPU to Snoop Them All: a Full-Band Bluetooth Low Energy Sniffer. In *Mediterranean Communication and Computer Networking Conference (MedComNet)*, 2020.

[59] Constitute Project. "Right to Privacy" on Constitute. https://www.constituteproject.org/search?lang=en&key=privacy, 2021. Accessed: 2017-12-12.

[60] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde. Analyzing Android Encrypted Network Traffic to Identify User Actions. In *IEEE Transactions on Information Forensics and Security (TIFS)*, 2015.

[61] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An Anonymous Messaging System Handling Millions of Users. In *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[62] H. Corrigan-Gibbs and B. Ford. Dissent: Accountable Anonymous Group Messaging. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2010.

[63] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford. Proactively Accountable Anonymous Messaging In Verdict. In *USENIX Security Symposium*, 2013.

[64] D. Cox. The Rise of Employee Health Tracking. https://www.bbc.com/worklife/article/20201110-the-rise-of-employee-health-tracking, 2020. Accessed: 2020-11-13.

[65] CRAWDAD. A Community Resource for Archiving Wireless Data At Dartmouth. http://crawdad.org/, 2016. Accessed: 2017-12-12.

[66] G. Danezis and R. Clayton. Introducing Traffic Analysis. In *Digital Privacy: Theory, Technologies, and Practices*, 2007.

[67] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *IEEE Symposium on Security and Privacy (S&P)*, 2003.

[68] G. Danezis and I. Goldberg. Sphinx: A Compact and Provably Secure Mix Format. In *IEEE Symposium on Security and Privacy (S&P)*, 2009.

[69] A. K. Das, P. H. Pathak, C.-N. Chuah, and P. Mohapatra. Uncovering Privacy Leakage In BLE Network Traffic of Wearable Fitness Trackers. In *International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2016.

[70] D. Das, S. Meiser, E. Mohammadi, and A. Kate. Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low latency - choose two. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[71] K. Dave, V. Varma, et al. Computational Advertising: Techniques for Targeting Relevant Ads. In *Foundations and Trends in Information Retrieval*, 2014.

[72] E. Debuf. Tools to do the Job: The ICRC's Legal Status, Privileges and Immunities. https://www.icrc.org/en/international-review/article/tools-do-job-icrcs-legal-status-privileges-and-immunities, 2016. Accessed: 2021-04-07.

[73] DEDIS. Kyber. https://github.com/dedis/kyber, 2020. Accessed: 2021-04-07.

[74] C. Delerablée. Identity-Based Broadcast Encryption with Constant Size Ciphertexts and Private Keys. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2007.

[75] DeterLab. Deterlab: Cyber-Defense Technology Experimental Research Laboratory. https://www.isi.deterlab.net, 2016. Accessed: 2021-04-07.

[76] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, IETF, 2008.

[77] R. Dingledine and N. Mathewson. Anonymity Loves Company: Usability and The Network Effect. In *WEIS*, 2006.

[78] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. Technical report, Naval Research Lab Washington DC, 2004.

[79] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani. Characterization of Encrypted and VPN Traffic Using Time-related. In *International Conference on Information Systems Security and Privacy (ICISSP)*, 2016.

[80] R. Dubin, A. Dvir, O. Pele, and O. Hadar. I Know What You Saw Last Minute - Encrypted HTTP Adaptive Video Streaming Title Classification. In *IEEE Transactions on Information Forensics and Security (TIFS)*, 2017.

[81] C. Dwork, A. Roth, et al. The Algorithmic Foundations of Differential Privacy. In *Foundations and Trends in Theoretical Computer Science*, 2014.

[82] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *IEEE Symposium on Security and Privacy*, 2012.

[83] Ellisys. Ellisys Bluetooth Vanguard Advanced All-in-one Bluetooth Analysis System. https://www.ellisys.com/products/bv1/, 2020. Accessed: 2020-09-20.

[84] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh. Express: Lowering The Cost of Metadata-Hiding Communication with Cryptographic Privacy. In *arXiv preprint 1911.09215*, 2019.

[85] Facebook, Inc. Facebook Ads: Online Advertising on Facebook. https://www.facebook.com/business/ads, 2021. Accessed: 2021-04-07.

[86] N. Fazio and I. M. Perera. Outsider-Anonymous Broadcast Encryption with Sublinear Ciphertexts. In *International Workshop on Public Key Cryptography*, 2012.

[87] J. Feigenbaum and B. Ford. Multiple Objectives of Lawful-surveillance Protocols. In *Cambridge International Workshop on Security Protocols*, 2017.

[88] U. Food and D. Administration. Content of Premarket Submissions for Management of Cybersecurity in Medical Devices. https://www.fda.gov/media/119933/download, 2018. Accessed: 2020-06-29.

[89] B. Ford, L. Barman, J.-P. Hubaux, I. Dacosta, and J. Feigenbaum. Systems and Method for Anonymous, Low-latencey, Tracking-Resistant Communications In a Networked Environment. Technical report, InfoScience EPFL, 2018.

[90] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. V. Randwyk, and D. Sicker. Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting. In *USENIX Security Symposium*, 2006.

[91] M. J. Freedman and R. Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2002.

[92] S. Frolov and E. Wustrow. The Use of TLS In Censorship Circumvention. In *Network and Distributed Systems Symposium (NDSS)*, 2019.

[93] J. Furukawa and K. Sako. An Efficient Scheme for Proving a Shuffle. In *Annual International Cryptology Conference*, 2001.

[94] P. Garcia, J. Van De Graaf, A. Hevia, and A. Viola. Beating The Birthday Paradox In Dining Cryptographer Networks. In *International Conference on Cryptology and Information Security in Latin America*, 2014.

[95] P. Garcia, J. Van De Graaf, G. Montejano, D. Riesco, N. Debnath, and S. Bast. Storage Optimization for Non-Interactive Dining Cryptographers (NIDC). In *International Conference on Information Technology New Generations (ITNG)*, 2015.

[96] C. Gentry and B. Waters. Adaptive Security In Broadcast Encryption Systems (With Short Ciphertexts). In *International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, 2009.

[97] Y. Gilad. Metadata-Private Communication for The 99%. In *Communications of the ACM (CACM)*, 2019.

[98] Y. Gluck, N. Harris, and A. Prado. Breach: Reviving the Crime Attack. http://breachattack.com/resources/BREACH%20-%20SSL,%20gone%20in%2030%20seconds.pdf, 2013. Accessed: 2021-04-07.

[99] S. Goel, M. Robson, M. Polte, and E. Sirer. Herbivore: A Scalable and Efficient Protocol for Anonymous Communication. Technical report, Cornell University, 2003.

[100] D. Goldschlag, M. Reed, and P. Syverson. Onion Routing for Anonymous and Private Internet Connections. In *Communications of the ACM (CACM)*, 1999.

[101] P. Golle and A. Juels. Dining Cryptographers Revisited. In *International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, 2004.

[102] J. Gong and T. Wang. Zero-Delay Lightweight Defenses Against Website Fingerprinting. In *USENIX Security Symposium*, 2020.

[103] X. Gong, N. Kiyavash, and N. Borisov. Fingerprinting Websites Using Remote Traffic Analysis. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2010.

[104] Google. Monkeyrunner. https://developer.android.com/studio/test/monkeyrunner/, 2020. Accessed: 2020-03-13.

[105] Google. Personalized Advertising. https://support.google.com/adspolicy/answer/143465, 2021. Accessed: 2021-04-07.

[106] Google. HTTPS Encryption on the Web . https://transparencyreport.google.com/https/overview?hl=en, 2021. Accessed: 2021-04-07.

[107] GreatScottGadgets. Ubertooth. https://github.com/greatscottgadgets/ubertooth, 2020. Accessed: 2020-11-09.

[108] B. Greschbach, G. Kreitz, and S. Buchegger. The Devil is In The Metadata — New Privacy Challenges In Decentralised Online Social Networks. In *IEEE International Conference on Pervasive Computing and Communications Workshops*, 2012.

[109] N. W. Group. IP encapsulating security payload (ESP). Technical report, IETF, 1998.

[110] N. W. Group. Provider provisioned virtual private network (VPN) terminology. Technical report, IETF, 2005.

[111] C. Gulcu and G. Tsudik. Mixing E-mail with Babel. In *Network and Distributed Systems Symposium (NDSS)*, 1996.

[112] K. Haataja and K. Hypponen. Man-in-the-Middle Attacks on Bluetooth: a Comparative Analysis, a Novel Attack, and Countermeasures. In *International Symposium on Communications, Control and Signal Processing (ISCCSP)*, 2008.

[113] K. Haataja and P. Toivanen. Practical Man-in-the-Middle Attacks Against Bluetooth Secure Simple Pairing. In *International Conference on Wireless Communications, Networking and Mobile Computing (WiCom)*, 2008.

[114] K. Haataja and P. Toivanen. Two Practical Man-in-the-Middle Attacks on Bluetooth Secure Simple Pairing and Countermeasures. In *IEEE Transactions on Wireless Communications*, 2010.

[115] D. Herrmann, R. Wendolsky, and H. Federrath. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with The Multinomial Naïve-bayes Classifier. In *ACM Workshop on Cloud Computing Security (CCSW)*, 2009.

[116] A. Hilts, C. Parsons, and J. Knockel. Every Step you Fake: A Comparative Analysis of Fitness Tracker Privacy and Security. In *Open Effect Report*, 2016.

[117] S. Hollister. Carriers Selling your Location to Bounty Hunters: it Was Worse than We Thought. https://www.theverge.com/2019/2/6/18214667/att-t-mobile-sprint-location-tracking-data-bounty-hunters, 2019. Accessed: 2021-02-12.

[118] A. Houmansadr, C. Brubaker, and V. Shmatikov. The Parrot is Dead: Observing Unobservable Network Communications. In *IEEE Symposium on Security and Privacy*, 2013.

[119] H.-C. Hsiao, T. H.-J. Kim, A. Perrig, A. Yamada, S. C. Nelson, M. Gruteser, and W. Meng. LAP: Lightweight Anonymity and Privacy. In *IEEE Symposium on Security and Privacy (S&P)*, 2012.

[120] J. Huang, W. Albazrqaoe, and G. Xing. BlueID: A Practical System for Bluetooth Device Identification. In *IEEE Conference on Computer Communications (INFOCOM)*, 2014.

[121] International Telecommunication Union. ITU-T G.114 - Amendment 2: New Appendix III – Delay Variation on Unshared Access Lines. https://www.itu.int/rec/T-REC-G.114-200911-I!Amd2/en, 2009. Accessed: 2021-04-07.

[122] M. Jakobsson. Flash Mixing. In *ACM Symposium on Principles of Distributed Computing (PODC)*, 1999.

[123] T. Jason Hiner. Understanding Snowden's Impact on IT... in 2 Minutes. https://www.techrepublic.com/blog/tech-sanity-check/video-understanding-snowdens-impact-on-it-in-2-minutes/, 2013. Accessed: 2021-04-07.

[124] M. Jeff Schultz. How Much Data is Created on the Internet Each Day? . https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/, 2019. Accessed: 2017-12-12.

[125] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2013.

[126] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright. Toward an Efficient Website Fingerprinting Defense. In *European Symposium on Research in Computer Security (ESORICS)*, 2016.

[127] G. Kalic, I. Bojic, and M. Kusek. Energy Consumption In Android Phones when Using Wireless Communication Technologies. In *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2012.

[128] J. Kelsey. Compression and Information Leakage of Plaintext. In *International Workshop on Fast Software Encryption*, 2002.

[129] D. Kesdogan, J. Egner, and R. Büschkes. Stop-and-Go-mixes Providing Probabilistic Anonymity In an Open System. In *International Workshop on Information Hiding*, 1998.

[130] A. Kharpal. Coronavirus: Israel, China, Singapore Use Surveillance Tech to Track Covid-19. https://www.cnbc.com/2020/03/27/coronavirus-surveillance-used-by-governments-to-fight-pandemic-privacy-concerns.html, 2020. Accessed: 2021-02-12.

[131] J. Korhonen and Y. Wang. Effect of Packet Size on Loss Rate and Delay In Wireless Links. In *IEEE Wireless Communications and Networking Conference (WCNC)*, 2005.

[132] A. Korolova and V. Sharma. Cross-App Tracking via Nearby Bluetooth Low Energy Devices. In *ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2018.

[133] A. Krasnova, M. Neikes, and P. Schwabe. Footprint Scheduling for Dining-Cryptographer Networks. In *International Conference on Financial Cryptography and Data Security*, 2016.

[134] C. Kuhn, M. Beck, S. Schiffner, E. Jorswieck, and T. Strufe. On Privacy Notions In Anonymous Communication. In *Privacy Enhancing Technologies Symposium (PETS)*, 2019.

[135] M. Kwet. In Stores, Secret Surveillance Tracks Your Every Move. https://www.nytimes.com/interactive/2019/06/14/opinion/bluetooth-wireless-tracking-privacy.html, 2019. Accessed: 2020-09-20.

[136] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford. Atom: Horizontally Scaling Strong Anonymity. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[137] A. Kwon, D. Lazar, S. Devadas, and B. Ford. Riffle: An Efficient Communication System with Strong Anonymity. In *Privacy Enhancing Technologies Symposium (PETS)*, 2016.

[138] A. Kwon, D. Lu, and S. Devadas. XRD: Scalable Messaging System with Cryptographic Privacy. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[139] L. Lamport et al. Paxos Made Simple. In *ACM Sigact News*, 2001.

[140] A. Langley. Pond. https://github.com/agl/pond, 2016. Accessed: 2021-04-07.

[141] D. Lazar, Y. Gilad, and N. Zeldovich. Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[142] D. Lazar, Y. Gilad, and N. Zeldovich. Yodel: Strong Metadata Security for Voice Calls. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2019.

[143] D. Lazar and N. Zeldovich. Alpenhorn: Bootstrapping Secure Communication without Leaking Metadata. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[144] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt. Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for VoIP Systems. In *ACM SIGCOMM Computer Communication Review*, 2015.

[145] S. Le Blond, D. Choffnes, W. Zhou, P. Druschel, H. Ballani, and P. Francis. Towards Efficient Traffic-Analysis Resistant Anonymity Networks. In *ACM SIGCOMM Computer Communication Review*, 2013.

[146] S. Le Blond, A. Cuevas, J. R. Troncoso-Pastoriza, P. Jovanovic, B. Ford, and J.-P. Hubaux. On Enforcing The Digital Immunity of a Large Humanitarian Organization. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[147] S. Le Blond, C. Zhang, A. Legout, K. Ross, and W. Dabbous. I Know Where You Are and What You Are Sharing: Exploiting P2P Communications to Invade Users' Privacy. In *ACM SIGCOMM Internet Measurement Conference*, 2011.

[148] M. Liberatore and B. N. Levine. Inferring The Source of Encrypted HTTP Connections. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2006.

[149] X. Liu, T. Chen, F. Qian, Z. Guo, F. X. Lin, X. Wang, and K. Chen. Characterizing Smartwatch Usage In The Wild. In *ACM Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2017.

[150] LiveRamp. LiveRamp Customer Onboarding. https://liveramp.com/our-platform/data-onboarding/, 2020. Accessed: 2020-03-05.

[151] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller. HoneyBadgerMPC and AsynchroMix: Practical Asynchronous Mpc and its Application to Anonymous Communication. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[152] X. Luo, P. Zhou, E. W. Chan, W. Lee, R. K. Chang, and R. Perdisci. HTTPOS: Sealing Information Leaks with Browser-Side Obfuscation of Encrypted Flows. In *Network and Distributed Systems Symposium (NDSS)*, 2011.

[153] D. Mantz, J. Classen, M. Schulz, and M. Hollick. InternalBlue - Bluetooth Binary Patching and Experimentation Framework. In *ACM Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2019.

[154] J. Martin, D. Alpuche, K. Bodeman, L. Brown, E. Fenske, L. Foppe, T. Mayberry, E. Rye, B. Sipes, and S. Teplov. Handoff All your Privacy - a Review of Apple's Bluetooth Low Energy Continuity Protocol. In *Privacy Enhancing Technologies Symposium (PETS)*, 2019.

[155] J. Mayer, P. Mutchler, and J. C. Mitchell. Evaluating The Privacy Properties of Telephone Metadata. In *Proceedings of the National Academy of Sciences*, 2016.

[156] J. McCarthy. TFL Introduces WiFi Tracking to Improve Ads. https://www.thedrum.com/news/2019/05/22/tfl-introduces-wifi-tracking-improve-ads, 2019. Accessed: 2020-03-05.

[157] H. Mohajeri Moghaddam, B. Li, M. Derakhshani, and I. Goldberg. Skypemorph: Protocol Obfuscation for Tor Bridges. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2012.

[158] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Anonymous Remailer. http://mixmaster.sourceforge.net/, 2003. Accessed: 2021-04-07.

[159] C. A. Neff. Verifiable Mixing (Shuffling) of Elgamal Pairs. In *VHTi Technical Document, VoteHere, Inc.*, 2003.

[160] L. Nguyen and R. Safavi-naini. Breaking and Mending Resilient Mix-Nets. In *Privacy Enhancing Technologies Symposium (PETS)*, 2003.

[161] K. Nikitin, L. Barman, W. Lueks, M. Underwood, J.-P. Hubaux, and B. Ford. Reducing Metadata Leakage from Encrypted Files and Communication with Purbs. In *Privacy Enhancing Technologies Symposium (PETS)*, 2019.

[162] K. Nikitin, L. Barman, W. Lueks, M. Underwood, J.-P. Hubaux, and B. Ford. Repository for "PURBs". https://github.com/dedis/purb, 2020. Accessed: 2021-05-01.

[163] R. Overdorf, M. Juarez, G. Acar, R. Greenstadt, and C. Diaz. How Unique is Your .onion?: An Analysis of The Fingerprintability of Tor Onion Services. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[164] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website Fingerprinting In Onion Routing Based Anonymization Networks. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2011.

[165] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 User Fingerprinting. In *ACM Conference on Mobile Computing and Networking (MobiCom)*, 2007.

[166] T. Peng, C. Leckie, and K. Ramamohanarao. Protection from Distributed Denial of Service Attacks Using History-Based IP Filtering. In *IEEE International Conference on Communications, . ICC'03.*, 2003.

[167] C. Perlich, B. Dalessandro, T. Raeder, O. Stitelman, and F. Provost. Machine Learning for Targeted Display Advertising: Transfer Learning In Action. In *Machine learning*, 2014.

[168] B. Pfitzmann. Breaking an Efficient Anonymous Channel. In *International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, 1995.

[169] R. C.-W. Phan and P. Mingard. Analyzing The Secure Simple Pairing In Bluetooth v4. 0. In *Wireless Personal Communications*, 2012.

[170] C. Phillips and S. Singh. CRAWDAD Dataset pdx/vwave (v. 2007-09-14). http://crawdad.org/pdx/vwave/20070914/wlan_pcap, 2007. Accessed: Traceset: wlan_pcap. 2021-04-07.

[171] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. The Loopix Anonymity System. In *USENIX Security Symposium*, 2017.

[172] M. A. Poletto and A. E. Dudfield. Architecture to Thwart Denial of Service Attacks. https://www.freepatentsonline.com/y2003/0204621.html, 2010. Accessed: 2021-04-07.

[173] M. S. Rahman, P. Sirinam, N. Mathews, K. G. Gangadhara, and M. Wright. Tik-Tok: The Utility of Packet Timing In Website Fingerprinting Attacks. In *Privacy Enhancing Technologies Symposium (PETS)*, 2020.

[174] A. Reed and B. Klimkowski. Leaky Streams: Identifying Variable Bitrate Dash Videos Streamed Over Encrypted 802.11n Connections. In *IEEE Consumer Communications & Networking Conference (CCNC)*, 2016.

[175] A. Reed and M. Kranch. Identifying HTTPS-Protected Netflix Videos In Real-time. In *ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2017.

[176] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. In *ACM Transactions on Information and System Security (TISSEC)*, 1998.

[177] M. Rennhard and B. Plattner. Introducing Morphmix: Peer-to-Peer Based Anonymous Internet Usage with Collusion Detection. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2002.

[178] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, IETF, 2018.

[179] E. Rescorla. A Readable Specification of TLS 1.3. https://www.davidwong.fr/tls13/, 2018. Accessed: 2021-04-07.

[180] RingRoad. Ring-Road: Leaking Sensitive Data in Security Protocols. http://www.ringroadbug.com/, 2017. Accessed: 2021-04-07.

[181] I. Ristić. HTTP Client Fingerprinting Using SSL Handshake Analysis. https://blog.ivanristic.com/2009/06/http-client-fingerprinting-using-ssl-handshake-analysis.html, 2009. Accessed: 2021-04-07.

[182] J. Rizzo and T. Duong. The Crime Attack. https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU, 2012. Accessed: 2021-04-07.

[183] P. Rogaway. Nonce-Based Symmetric Encryption. In *International Workshop on Fast Software Encryption*, 2004.

[184] J. Ruge, J. Classen, F. Gringoli, and M. Hollick. Frankenstein: Advanced Wireless Fuzzing to Exploit New Bluetooth Escalation Targets. In *USENIX Security Symposium*, 2020.

[185] M. Ryan. Bluetooth: With Low Energy Comes Low Security. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2013.

[186] B. Saltaformaggio, H. Choi, K. Johnson, Y. Kwon, Q. Zhang, X. Zhang, D. Xu, and J. Qian. Eavesdropping on Fine-Grained User Activities within Smartphone Apps Over Encrypted Network Traffic. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2016.

[187] J. Sankey and M. Wright. Dovetail: Stronger Anonymity In Next-Generation Internet Routing. In *Privacy Enhancing Technologies Symposium (PETS)*, 2014.

[188] D. Schatz, M. Rossberg, and G. Schaefer. Hydra: Practical Metadata Security for Contact Discovery, Messaging, and Dialing. In *International Conference on Information Systems Security and Privacy (ICISSP)*, 2021.

[189] R. Schuster, V. Shmatikov, and E. Tromer. Beauty and The Burst: Remote Identification of Encrypted Video Streams. In *USENIX Security Symposium*, 2017.

[190] A. Segal, J. Feigenbaum, and B. Ford. Privacy-Preserving Lawful Contact Chaining. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2016.

[191] A. Segal, B. Ford, and J. Feigenbaum. Catching Bandits and only Bandits: Privacy-Preserving Intersection Warrants for Lawful Surveillance. In *USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2014.

[192] S. Sengupta, H. Gupta, N. Ganguly, B. Mitra, P. De, and S. Chakraborty. CRAWDAD Dataset iitkgp/apptraffic (v. 2015-11-26). http://crawdad.org/iitkgp/apptraffic/20151126/apptraffictraces, 2015. Accessed: Traceset: apptraffictraces. 2021-04-07.

[193] SequoiaPGP Documentation. SequoiaPGP - Padmé. https://docs.sequoia-pgp.org/sequoia_openpgp/serialize/stream/padding/fn.padme.html, 2021. Accessed: 2021-04-07.

[194] D. Shah et al. Gossip Algorithms. In *Foundations and Trends in Networking*, 2009.

[195] R. K. Sheshadri and D. Koutsonikolas. On Packet Loss Rates In Modern 802.11 Networks. In *IEEE INFOCOM -IEEE Conference on Computer Communications*, 2017.

[196] R. Shields. London's Black Cabs are to be Fitted with Bluetooth Enabled Beacon Technology. https://www.thedrum.com/news/2015/09/07/london-s-black-cabs-are-be-fitted-bluetooth-enabled-beacon-technology-, 2015. Accessed: 2020-09-20.

[197] Signal. Signal. https://www.signal.org/, 2021. Accessed: 2021-04-07.

[198] J. Sipilä. Patients in Finland Blackmailed After Therapy Records were Stolen by Hackers. https://edition.cnn.com/2020/10/27/tech/finland-therapy-patients-blackmailed-data-breach-intl/index.html, 2020. Accessed: 2021-04-07.

[199] P. Sirinam, M. Imani, M. Juarez, and M. Wright. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.

[200] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright. Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-shot Learning. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[201] J.-P. Smith, P. Mittal, and A. Perrig. Website Fingerprinting In The Age of QUIC. In *Privacy Enhancing Technologies Symposium (PETS)*, 2021.

[202] Sodera. Sodera Series of Bluetooth Analyzers. https://fte.com/products/sodera.aspx, 2020. Accessed: 2020-10-19.

[203] D. Spill and A. Bittau. BlueSniff: Eve Meets Alice and Bluetooth. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2007.

[204] V. Srinivasan, J. Stankovic, and K. Whitehouse. Protecting your Daily In-Home Activity Information from a Wireless Snooping Attack. In *International Conference on Ubiquitous Computing (UbiComp)*, 2008.

[205] Statista. Forecast Unit Shipments of Wearable Devices Worldwide from 2017 to 2019 and in 2022 (in Million units), by Category. https://www.statista.com/statistics/385658/electronic-wearable-fitness-devices-worldwide-shipments/, 2020. Accessed: 2020-10-28.

[206] A. A. Tabassam and S. Heiss. Bluetooth Clock Recovery and Hop Sequence Synchronization Using Software Defined Radios. In *IEEE Region 5 Conference*, 2008.

[207] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinez. Robust Smartphone App Identification via Encrypted Network Traffic Analysis. In *IEEE Transactions on Information Forensics and Security (TIFS)*, 2017.

[208] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. AppScanner: Automatic Fingerprinting of Smartphone Apps from Encrypted Network Traffic. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016.

[209] The Tor Project. Tor at the Heart: Bridges and Pluggable Transports. https://blog.torproject.org/tor-heart-bridges-and-pluggable-transports, 2016. Accessed: 2021-04-07.

[210] The Tor Project. Tor FAQ. https://2019.www.torproject.org/docs/faq.html.en#EntryGuards, 2021. Accessed: 2021-04-07.

[211] M. Tibouchi. Elligator Squared: Uniform Points on Elliptic Curves of Prime Order as Uniform Random Strings. In *International Conference on Financial Cryptography and Data Security*, 2014.

[212] J. P. Timpanaro, C. Isabelle, and F. Olivier. Monitoring The I2P Network. In *INRIA*, 2011.

[213] Tizen. sdb. https://developer.tizen.org/development/tizen-studio/web-tools/running-and-testing-your-app/sdb, 2020. Accessed: 2020-11-104.

[214] Trend Micro. What do Hackers do with Your Stolen Identity? https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/what-do-hackers-do-with-your-stolen-identity, 2017. Accessed: 2021-04-07.

[215] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky. Packet-level signatures for smart home devices. *Network and Distributed Systems Symposium (NDSS)*, 2020.

[216] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich. Stadium: A Distributed Metadata-Private Messaging System. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[217] U.S. State Department. Privileges and Immunities. https://www.state.gov/privileges-and-immunities/, 2018. Accessed: 2021-04-07.

[218] T. Valverde. Bad Life Advice - Replay Attacks Against HTTPS. http://blog.valverde.me/2015/12/07/bad-life-advice/, 2015. Accessed: 2021-04-07.

[219] J. Van De Graaf. Anonymous One-Time Broadcast Using Non-Interactive Dining Cryptographer Nets with Applications to Voting. In *Towards Trustworthy Election, Springler*, 2010.

[220] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2015.

[221] VoIP-Info. VoIP QoS Requirements. https://www.voip-info.org/QoS, 2017. Accessed: 2021-04-07.

[222] G. Vranken. HTTPS Bicycle Attack. https://guidovranken.com/2015/12/30/https-bicycle-attack/, 2015. Accessed: 2021-04-07.

[223] J. Wang, F. Hu, Y. Zhou, Y. Liu, H. Zhang, and Z. Liu. BlueDoor: Breaking The Secure Information Flow via BLE Vulnerability. In *ACM Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2020.

[224] Q. Wang, X. Gong, G. T. Nguyen, A. Houmansadr, and N. Borisov. Censorspoofer: Asymmetric Communication Using IP Spoofing for Censorship-Resistant Web Browsing. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2012.

[225] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective Attacks and Provable Defenses for Website Fingerprinting. In *USENIX Security Symposium*, 2014.

[226] T. Wang and I. Goldberg. Improved Website Fingerprinting on Tor. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2013.

[227] T. Wang and I. Goldberg. On Realistically Attacking Tor with Website Fingerprinting. In *Privacy Enhancing Technologies Symposium (PETS)*, 2016.

[228] T. Wang and I. Goldberg. Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks. In *USENIX Security Symposium*, 2017.

[229] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh. Stegotorus: a Camouflage Proxy for The Tor Anonymity System. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2012.

[230] WhatsApp. About End-to-End Encryption. https://faq.whatsapp.com/general/security-and-privacy/end-to-end-encryption/?lang=en, 2021. Accessed: 2021-04-07.

[231] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monrose. Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on fon-iks. In *IEEE Symposium on Security and Privacy (S&P)*, 2011.

[232] P. Winter, T. Pulls, and J. Fuss. Scramblesuit: A Polymorphic Network Protocol to Circumvent Censorship. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2013.

[233] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent In Numbers: Making Strong Anonymity Scale. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

[234] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Scalable Anonymous Group Communication In The Anytrust Model. In *Naval Research Lab Washington DC*, 2012.

[235] F.-L. Wong, F. Stajano, and J. Clulow. Repairing The Bluetooth Pairing Protocol. In *International Workshop on Security Protocols*, 2005.

[236] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson. Language Identification of Encrypted VoIP Traffic: Alejandra y Roberto or Alice and Bob? In *USENIX Security Symposium*, 2007.

[237] C. V. Wright, S. E. Coull, and F. Monrose. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In *Network and Distributed Systems Symposium (NDSS)*, 2009.

[238] M. K. Wright, M. Adler, B. N. Levine, and C. Shields. The Predecessor Attack: An Analysis of a Threat to Anonymous Communications Systems. In *ACM Transactions on Information and System Security (TISSEC)*, 2004.

[239] J. Wu, Y. Nan, V. Kumar, D. J. Tian, A. Bianchi, M. Payer, and D. Xu. BLESA: Spoofing Attacks Against Reconnections In Bluetooth Low Energy. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2020.

[240] Q. Xu, R. Zheng, W. Saad, and Z. Han. Device Fingerprinting In Wireless Networks: Challenges and Opportunities. In *IEEE Communications Surveys & Tutorials*, 2016.

[241] Y. Yang, N. Liu, S.-L. Wong, and Q. Lui. China, Coronavirus and Surveillance: the Messy Reality of Personal Data. https://www.ft.com/content/760142e6-740e-11ea-95fe-fcd274e920ca, 2020. Accessed: 2021-02-12.

[242] D. Yanofsky. Google can Still Use Bluetooth to Track your Android Phone when Bluetooth is Turned off. https://qz.com/1169760/phone-data/, 2018. Accessed: 2020-03-05.

[243] B. Zantout, R. Haraty, et al. I2P Data Communication System. In *Proceedings of ICN*, 2011.

[244] F. Zhang, W. He, X. Liu, and P. G. B. Bridges. Inferring Users' Online Activities Through Traffic Analysis. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2011.

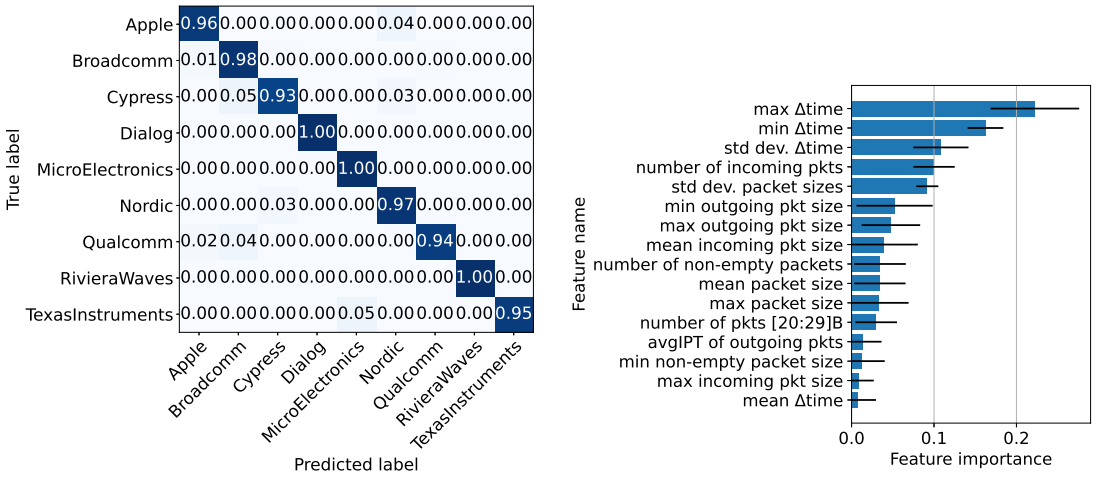
[245] Y. Zhang, J. Weng, R. Dey, Y. Jin, Z. Lin, and X. Fu. Breaking Secure Pairing of Bluetooth Low Energy Using Downgrade Attacks. In *USENIX Security Symposium*, 2020.

[246] P. R. Zimmermann. The Official PGP User's Guide. In *MIT Press*, 1995.

[247] C. Zuo, H. Wen, Z. Lin, and Y. Zhang. Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

A. Appendix: Wearable Devices (Chapter 2)

A.1 Chipset Fingerprinting



(a) Normalized confusion matrix per true label.

(b) Feature importance.

Figure A.1 – Chipset identification (Classic & LE)

The classifier used for device identification (§2.5) fingerprints a combination of the hardware, the firmware, the OS, and the applications installed. To specifically target the hardware (*i.e.*, chipset manufacturer), we reproduce the same experiment, this time using the Bluetooth chipset manufacturer as the classifier's label. Our goal is to investigate if our classifier learns common patterns from the encrypted traffic recorded by devices that have the same chipset manufacturer, *e.g.*, `Samsung Galaxy Watch` and `Huawei Watch 2`, which both have a `Broadcom` Bluetooth chip or `Mi Band 2, 3` and 4 all equipped with a Dialog Semiconductor chip. Once again, we find that the classifier's performance is high (96% precision/recall/F1 score, Figure A.1a, Table A.3), which indicates that there exist stable communication patterns across chipsets. We remark that the limited sample size (*i.e.*, $1-3$ devices per chipset manufacturer) limits the generalization of our conclusions; further analysis is needed in this specific direction. Nonetheless, this result demonstrates the robustness of our methodology, as our earlier results show that our device-identification classifier can distinguish between devices from the same vendor.

A.2 Dataset Aging

We now evaluate the effect of dataset aging, that is, the loss of accuracy incurred as the adversary uses older datasets. We perform an experiment in which we collect data from applications over 32 days; then, we measure the classifier accuracy when classifying each day's samples with the model trained on the data collected at day 0. We use the 39 high-volume applications presented in Section 2.6.2, from which we exclude 6 applications that were not

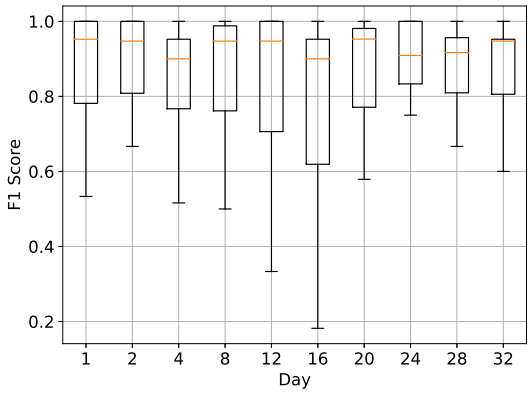
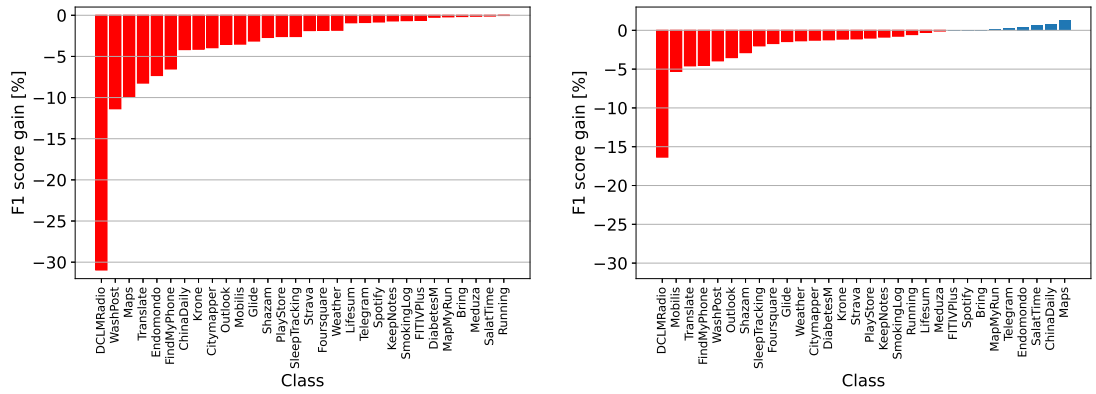


Figure A.1 – Training on day 0, testing on day i.



(a) Evolution of the F1 score per class, averaged over 32 days.

(b) Evolution of the F1 score per class, averaged over 29 days, training over the initial three days.

Figure A.2 – Effect of dataset aging on accuracy.

successfully automated and did not produce traces over the multiple days of this experiment. We also exclude 3 applications that stopped communicating data after an update over the month. Figure A.1 shows a box-plot of the F1 score per application over the duration of the experiment. First, we observe that the median F1 score is stable through the month, with small variations between 90% and 95%. The overall performance of the classifier did not vary significantly, even when using traces that are a month old. We do observe that the F1 score of some classes varies through the month (*e.g.,* on days 12 and 16, the lower whiskers indicate that some apps were misclassified more than usual). We investigate the F1 score per class, averaged over the month (Figure A.2a). We see that the different classes have variable F1 scores, in particular with one application being harder to classify: `DCLMRadio`. This a web radio that loads a large amount of content updated daily. It is unsurprising that training on a single day is not representative of the whole month. Nonetheless, other applications are still classified with high accuracy after 32 days.

Finally, to improve the classification, we also experiment with training the classifier using the data of several days. We use the data collected during the initial three days (instead of just the first), while keeping the same amount of training data (corresponding to a single day). We observe that training on just a few days leads to a performance increase of 3% mean accuracy (from 92% to 95%) over the month. The F1 score of `DCLMRadio`, the worst class, also increases by 14 percentage point (from 68% to 82%; Appendix, Figure A.2b). This suggests that an adversary does not need a fresh dataset to perform the attack. Thus, this lowers the cost of the attack by reducing the amount of training needed and by facilitating dataset reuse.

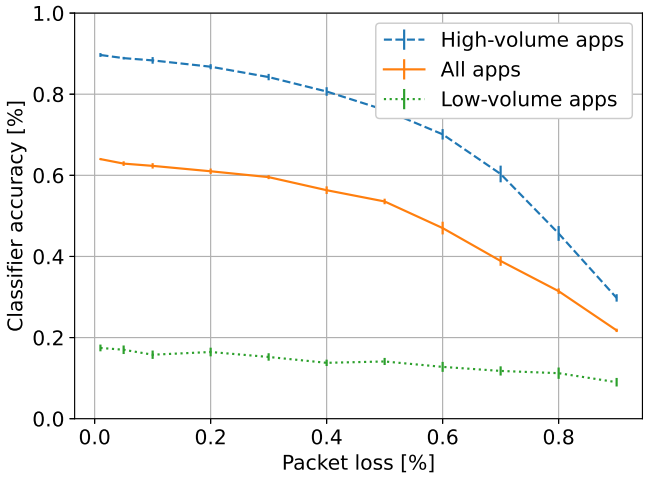


Figure A.1 – Packet loss rate versus classifier accuracy for application identification (“deep” experiment).

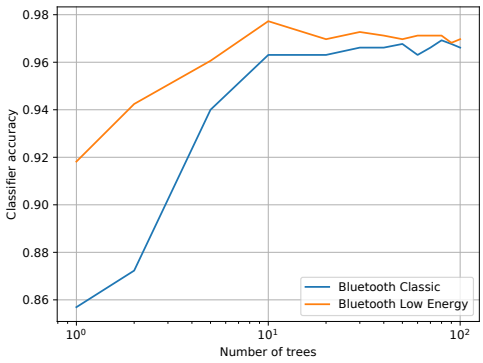
A.3 Impact of Packet Loss on the Attacks

In our experiments, we use a high-end sniffer that is co-located with the target devices. In this configuration, the sniffer has close to 100% packet capture rate. In practice, lower-end sniffers will suffer packet loss. Collisions from other devices and the distance between the target and the eavesdropper also increase loss. We briefly investigate how the attack accuracy varies with degraded capture conditions.

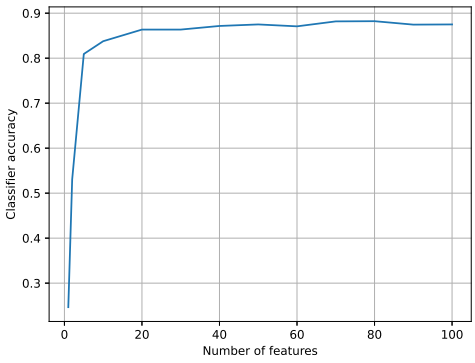
First, we decouple the attacker accuracy, the packet loss and the capture conditions, and we explore the attack accuracy versus the loss rate only. This loss can stem from many real-life parameters (distance, noise, multipath interference, the quality of the eavesdropping device, etc.). As a generic approach, we study the effect of uniform packet loss on our dataset. We simulate packet loss on the captured traces and re-run the application identification (“deep”) experiment (§2.6.2). We apply a uniform packet loss by dropping individual packets with a given probability. We then use the methodology already presented (we split the traces into train/test and compute the average classification accuracy). The experiment is repeated 10 times per loss rate. We observe (Appendix, Figure A.1) that even with 50% packet loss, the loss in accuracy is only 10 percentage points, with the mean accuracy dropping from 64% to 54%. For high-volume apps, the mean accuracy drops from 90% to 77%. This experiment indicates that the approach is robust to packet losses: even when missing every other packet, the attacker is able to classify with significant accuracy.

Due to the difficulty of relating the various capture conditions to the loss rate, we only discuss experimental results generated by Albazrqaoe *et al.* with the most common inexpensive Bluetooth sniffers: The authors describe how a single Ubertooth, when placed 10m away from the target device and in the presence of significant 802.11 interference [4], has between 25% and 50% packet loss. In similar conditions, a BlueEar sniffer (composed of two synchronized Ubertooth) maintains packet losses below 10% [4]. This observation suggests that the attack could also be performed with inexpensive Bluetooth sniffers.

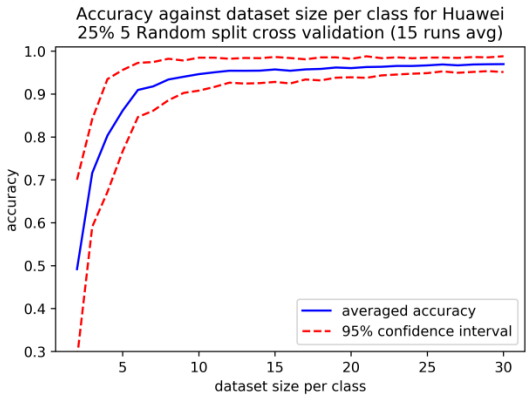
A.4 Additional Figures



(a) Number of trees, device identification (§2.5).

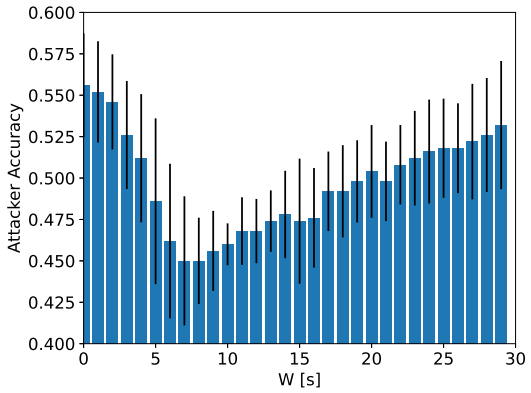


(b) Number of features kept by the Recursive Feature Elimination (RFE), device identification (§2.5).

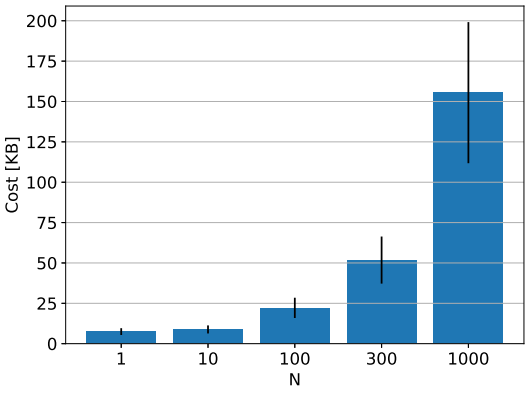


(c) Number of samples per class, application identification ("deep" experiment, §2.6.2).

Figure A.1 – Choice of parameters versus mean classifier accuracy.

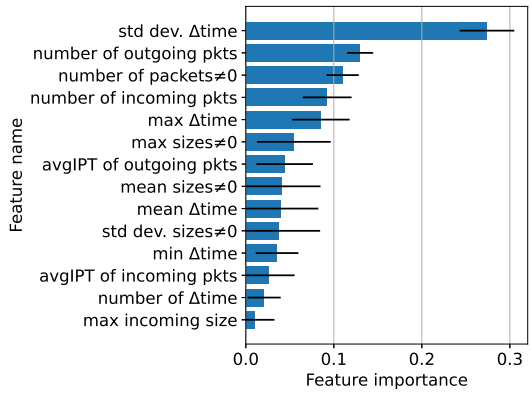


(a) Mean W of the Rayleigth distribution with respect to the attacker accuracy (lower is better).

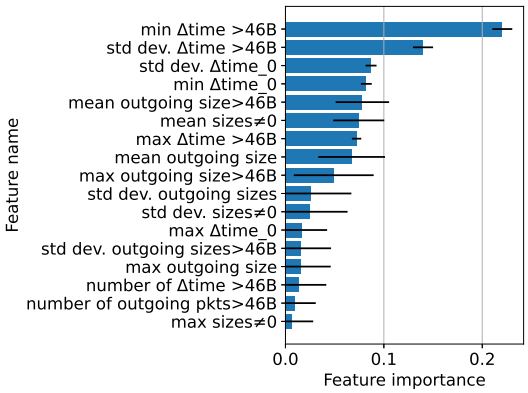


(b) Number of dummies N with respect to the cost of the defense. The cost is averaged per sample.

Figure A.2 – Parameter choices for `add_dummies`, illustrated here with the samples for application identification ("deep" experiment). We select $W = 6$s and $N = 300$

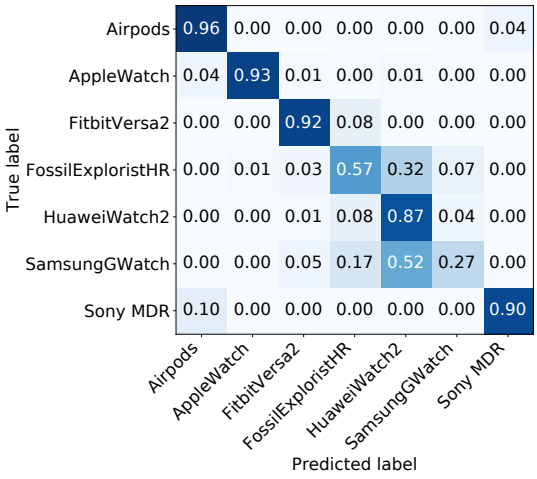


(a) Feature importance when attacking `pad`-defended traces, device identification, Bluetooth LE.

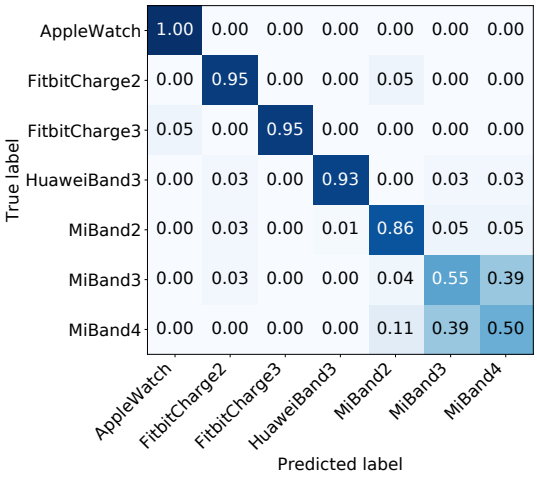


(b) Feature importance for action identification against `delay_group`-defended traces, `DiabetesM`.

Figure A.3 – Feature importance when attacking defended traces, two scenarios.

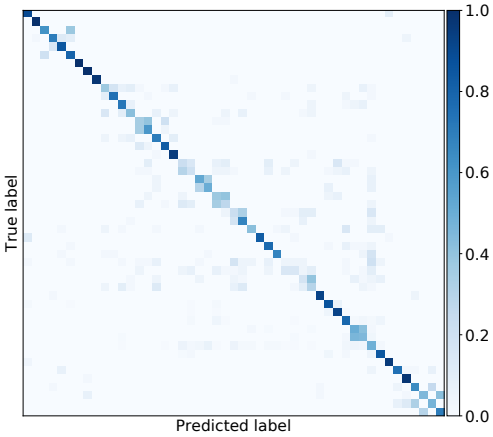


(a) Bluetooth Classic.

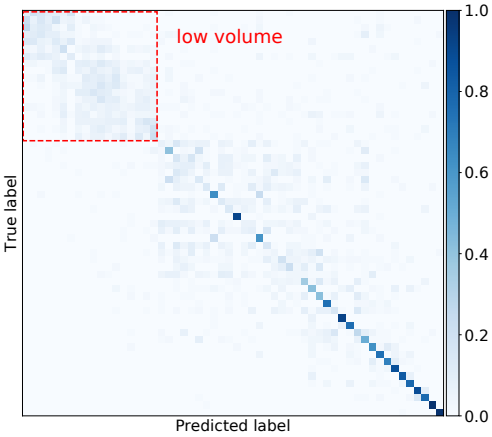


(b) Bluetooth Low Energy.

Figure A.4 – Normalized confusion matrices per true label, device identification against `add_dummies`-defended traces.



(a) Action identification ("wide" experiment on all wearables).



(b) Application identification ("deep" experiment).

Figure A.5 – Normalized confusion matrices per true label. Action and application identification against `add_dummies`-defended traces. The matrices are sorted by increasing median transmitted volume.

A.5 Tables

Classifier performance for Device identification.

Table A.1 – Bluetooth Classic.

Label	Precision	Recall	F1-score
Airpods	0.83	0.9	0.87
AppleWatch	0.98	0.93	0.96
FitbitVersa2	1.0	1.0	1.0
FossilExploristHR	0.96	0.98	0.97
HuaweiWatch2	0.98	0.98	0.98
SamsungGWatch	1.0	0.98	0.99
Sony MDR	0.9	0.86	0.88
Average	0.96	0.96	0.96

Table A.2 – Bluetooth LE.

Label	Precision	Recall	F1-score
AppleWatch	0.99	1.0	1.0
FitbitCharge2	1.0	1.0	1.0
FitbitCharge3	1.0	0.95	0.97
HuaweiBand3	1.0	1.0	1.0
MiBand2	0.93	0.94	0.93
MiBand3	0.88	0.98	0.92
MiBand4	0.99	0.86	0.92
Average	0.97	0.97	0.97

Classifier performances.

Table A.3 – Chipset identification.

Label	Precision	Recall	F1-score
Apple	0.9	0.94	0.92
Broadcomm	0.95	0.98	0.96
Cypress	0.98	0.98	0.98
Dialog	0.98	0.98	0.98
MicroElectronics	1.0	1.0	1.0
Qualcomm	0.97	0.88	0.92
RivieraWaves	0.98	1.0	0.99
Average	0.96	0.96	0.96

Table A.4 – Action identification within DiabetesM.

Label	Precision	Recall	F1-score
Add Calorie	0.44	0.46	0.45
Add Carbs	0.87	0.9	0.89
Add Fat	0.77	0.81	0.79
Add Glucose	0.85	0.91	0.88
Add Insulin	0.9	0.95	0.92
Add Proteins	0.37	0.28	0.32
Average	0.7	0.7	0.7

Table A.5 – 34 "high-volume" applications common between the two pairs of device Huawei Watch 2 - Pixel 2 and Fossil Q Explorist HR - Nexus 5.

Bring, Calm, ChinaDaily, Citymapper, DCLMRadio, DiabetesM, Endomondo, FITIVPlus, FindMyPhone, Fit, FitBreathe, FitWorkout, FoursquareCityGuide, Glide, KeepNotes, Krone, Lifesum, MapMyRun, Maps, Meduza, Mobills, Outlook, PlayStore, Running, SalatTime, Shazam, SleepTracking, SmokingLog, Spotify, Strava, Telegram, Translate, WashPost, Weather

Table A.6 – Applications and Actions used for the long-run captures.

Applications	AppInTheAir, Bring, Calm, ChinaDaily, Citymapper, DCLMRadio, DiabetesM, Endomondo, FITIVPlus, FindMyPhone, FoursquareCityGuide, Glide, KeepNotes, Krone, Lifesum, MapMyRun, Maps, Meduza, Mobills, Outlook, PlayStore, Qardio, Running, SalatTime, Shazam, SleepTracking, SmokingLog, Spotify, Strava, Telegram, Translate, WashPost, Weather.
Actions	DiabetesM_AddCalorie, DiabetesM_AddCarbs, DiabetesM_AddFat, DiabetesM_AddGlucose, DiabetesM_AddInsulin, DiabetesM_AddProteins, Endomondo_BrowseMap, Endomondo_Running, FoursquareCityGuide_Coffees, FoursquareCityGuide_Leisure, FoursquareCityGuide_NightLife, FoursquareCityGuide_Restaurants, FoursquareCityGuide_Shopping, HealthyRecipes_SearchRecipe, Lifesum_AddFood, Lifesum_AddWater, PlayStore_Browse.

Table A.7 – Classifier performance for Action identification, "Wide" experiment on all wearables.

Label	Precision	Recall	F1-score
Airpods_GooglePlayMusic_Play	0.87	0.96	0.91
AppleWatch_ECG_Sync	0.83	1.0	0.91
AppleWatch_Kaia_Workout	1.0	1.0	1.0
AppleWatch_MapMyRun_Workout	0.79	0.75	0.77
AppleWatch_Map_Browse	0.83	0.85	0.84
AppleWatch_Music_Skip	1.0	1.0	1.0
AppleWatch_PhotoApp_LiveStream	0.98	1.0	0.99
FitbitCharge2_Fitbit_Sync	0.98	1.0	0.99
FitbitCharge3_Fitbit_Sync	1.0	0.98	0.99
FossilExploristHR_EndomondoApp_Running	0.66	0.68	0.67
FossilExploristHR_EndomondoApp_Walking	0.74	0.78	0.76
FossilExploristHR_FITIVApp_Running	0.71	0.75	0.73
FossilExploristHR_FITIVApp_Walking	0.71	0.75	0.73
FossilExploristHR_MapMyRun_Running	0.76	0.8	0.78
FossilExploristHR_MapMyRun_Walking	0.77	0.75	0.76
FossilExploristHR_NoApp_EmailReceived	0.83	0.85	0.84
FossilExploristHR_NoApp_PhoneCallMissed	0.98	1.0	0.99
FossilExploristHR_NoApp_SmsReceived	0.95	0.98	0.96
GalaxyWatch_EndomondoApp_Running	0.42	0.45	0.43
GalaxyWatch_EndomondoApp_Walking	0.32	0.32	0.32
GalaxyWatch_FITIVApp_Running	0.61	0.57	0.59
GalaxyWatch_FITIVApp_Walking	0.65	0.65	0.65
GalaxyWatch_MapMyRun_Running	0.47	0.57	0.52
GalaxyWatch_MapMyRun_Walking	0.31	0.35	0.33
GalaxyWatch_MyFitnessPalApp_CaloriesAdd	0.84	0.78	0.81
GalaxyWatch_MyFitnessPalApp_WaterAdd	0.73	0.82	0.78
GalaxyWatch_NoApp_EmailReceived	0.83	0.6	0.7
GalaxyWatch_NoApp_PhoneCallMissed	1.0	0.9	0.95
GalaxyWatch_NoApp_PhotoTransfer	0.79	0.92	0.85
GalaxyWatch_NoApp_SmsReceived	0.85	0.82	0.84
GalaxyWatch_PearApp_Running	0.47	0.35	0.4
GalaxyWatch_PearApp_Walking	0.58	0.52	0.55
GalaxyWatch_SamsungHealthApp_Running	0.58	0.55	0.56
GalaxyWatch_SamsungHealthApp_Walking	0.67	0.55	0.6
HuaweiBand3_HuaweiApp_Sync	1.0	1.0	1.0
HuaweiWatch2_Endomondo_BrowseMap	0.94	0.92	0.93
HuaweiWatch2_Endomondo_Running	0.98	0.97	0.97
HuaweiWatch2_HealthyRecipes_SearchRecipe	0.92	0.93	0.93
HuaweiWatch2_Lifesum_AddFood	0.93	0.95	0.94
HuaweiWatch2_Lifesum_AddWater	0.93	1.0	0.96
HuaweiWatch2_NoApp_NoAction	0.81	0.79	0.8
HuaweiWatch2_PlayStore_Browse	0.94	0.85	0.89
MDR_GooglePlayMusic_Play	0.96	0.88	0.92
MiBand2_MiApp_Sync	1.0	0.95	0.97
MiBand2_MiApp_Walking	0.88	0.95	0.92
MiBand3_MiApp_Sync	1.0	1.0	1.0
MiBand3_MiApp_Walking	0.91	0.78	0.84
MiBand4_MiApp_Sync	0.93	0.98	0.95
MiBand4_MiApp_Walking	0.88	0.9	0.89
Average	0.82	0.82	0.82

Table A.8 – Details of transmitted volumes for the 18 “low-volume” apps over 40 recorded samples. NoApp corresponds to OS communications.

App	Median [B]	Std dev. [B]
Reminders	0.0	23660.5
Battery	0.0	239.6
DuaKhatqmAlQuran	11.0	43168.3
WearCasts	37.0	398149.0
DailyTracking	44.0	326.9
ASB	75.0	3511.6
NoApp	96.5	3949.5
HeartRate	104.0	21824.4
Workout	119.0	5108.4
AthkarOfPrayer	120.0	4835.5
Alarm	122.5	8891.4
GooglePay	127.0	2762.1
Flashlight	152.5	4176.7
Phone	154.5	2319.2
PlayMusic	156.0	16294.5
HealthyRecipes	167.5	3104.7
Sleep	171.5	14460.4
Medisafe	194.0	8802.6

Table A.9 – Details of transmitted volumes for the 38 “high-volume” apps over 40 samples.

App	Median [kB]	Std dev. [kB]
SalatTime	0.6	5.5
MapMyFitness	0.6	2.6
Citymapper	1.1	3.4
Calm	1.2	8.3
Outlook	1.4	3.4
DiabetesM	1.4	2.5
SmokingLog	1.5	46.7
MapMyRun	1.8	8.1
SleepTracking	1.9	4.4
Mobills	2.1	1.5
Fit	2.1	8.4
Weather	2.6	9.4
Running	2.9	8.0
FitWorkout	3.3	144.6
FitBreathe	3.3	3.4
FoursquareCityGuide	3.8	7.0
Glide	4.8	5.2
Translate	5.8	5.9
Shazam	6.2	41.9
Qardio	6.5	7.1
Krone	6.8	13.2
KeepNotes	7.0	4.8
FindMyPhone	8.0	12.2
Telegram	8.7	3.7
Strava	10.6	4.7
DCLMRadio	16.6	15.6
Lifesum	17.8	11.0
Endomondo	17.9	9.7
PlayStore	18.5	24.3
Maps	18.7	62.8
AppInTheAir	26.4	14.4
Bring	34.5	7.9
Spotify	38.8	9.8
Meduza	40.9	13.9
FITIVPlus	48.9	11.7
ChinaDaily	55.1	8.3
WashPost	120.1	14.2
Camera	598.0	141.5

Classifier performance for App identification, Huawei Watch.

Table A.10 – 18 “low-volume” applications.

Label	Precision	Recall	F1-score
Battery	0.21	0.31	0.25
Reminders	0.14	0.25	0.18
DuaKhatqmAl.	0.08	0.06	0.07
WearCasts	0.26	0.18	0.21
DailyTracking	0.2	0.15	0.17
ASB	0.25	0.28	0.26
NoApp	0.06	0.02	0.04
HeartRate	0.17	0.15	0.16
Workout	0.11	0.1	0.11
AthkarOfPrayer	0.09	0.09	0.09
Alarm	0.06	0.05	0.05
GooglePay	0.08	0.09	0.08
Flashlight	0.13	0.14	0.13
Phone	0.34	0.34	0.34
PlayMusic	0.2	0.2	0.2
HealthyRecipes	0.11	0.12	0.12
Sleep	0.21	0.24	0.22
Medisafe	0.32	0.32	0.32
Average	0.17	0.17	0.17

Table A.11 – 38 “high-volume” applications

Label	Precision	Recall	F1-score
SalatTime	1.0	1.0	1.0
MapMyFitness	1.0	0.96	0.98
Calm	0.94	0.96	0.95
Citymapper	0.94	0.96	0.95
DiabetesM	0.94	0.95	0.94
Outlook	0.96	0.92	0.94
SmokingLog	0.91	0.79	0.85
Fit	0.82	0.88	0.85
Running	0.78	0.72	0.75
MapMyRun	1.0	0.94	0.97
SleepTracking	0.99	0.92	0.95
Weather	0.76	0.69	0.72
Mobills	0.95	0.92	0.94
FitBreathe	0.96	0.99	0.98
Foursquare	0.91	0.94	0.93
FitWorkout	0.84	0.8	0.82
Glide	0.88	0.94	0.91
Translate	0.94	0.91	0.92
Qardio	0.95	0.95	0.95
Krone	0.91	0.84	0.87
FindMyPhone	0.81	0.84	0.82
KeepNotes	0.95	0.88	0.91
Shazam	0.95	0.86	0.9
Strava	0.86	0.79	0.82
Telegram	0.95	0.92	0.94
Maps	0.74	0.79	0.76
Endomondo	0.78	0.84	0.81
DCLMRadio	0.96	0.95	0.96
Lifesum	0.8	0.88	0.84
PlayStore	0.8	0.88	0.83
AppInTheAir	0.82	0.94	0.88
Bring	0.82	0.94	0.88
Spotify	0.96	0.99	0.98
Meduza	0.9	0.91	0.91
FITIVPlus	0.99	0.99	0.99
ChinaDaily	0.96	0.91	0.94
WashPost	0.9	1.0	0.95
Camera	1.0	1.0	1.0
Average	0.9	0.9	0.9

Classifier performance when transferring the model between pairs of devices.

Table A.12 – Train on Huawei-Pixel. Test on Fossil-Nexus.

Label	Precision	Recall	F1-score
Bring	1.0	1.0	1.0
Calm	0.96	0.86	0.91
ChinaDaily	0.96	0.89	0.93
Citymapper	0.74	1.0	0.85
DCLMRadio	0.91	0.71	0.8
DiabetesM	0.97	1.0	0.98
Endomondo	0.78	1.0	0.88
FITIVPlus	0.88	1.0	0.93
FindMyPhone	0.71	0.17	0.28
Fit	0.0	0.0	0.0
FitBreathe	0.3	0.93	0.46
FitWorkout	0.0	0.0	0.0
Foursquare	1.0	0.86	0.92
Glide	0.85	1.0	0.92
KeepNotes	0.71	0.96	0.82
Krone	1.0	0.62	0.77
Lifesum	0.84	0.96	0.9
MapMyRun	1.0	1.0	1.0
Maps	0.95	0.62	0.75
Meduza	0.85	0.97	0.9
Mobills	1.0	0.79	0.88
Outlook	1.0	0.96	0.98
PlayStore	0.33	0.11	0.16
Running	0.96	0.89	0.93
SalatTime	0.57	0.96	0.72
Shazam	1.0	0.82	0.9
SleepTracking	0.96	0.86	0.91
SmokingLog	0.96	0.89	0.93
Spotify	1.0	1.0	1.0
Strava	1.0	0.86	0.92
Telegram	1.0	0.97	0.98
Translate	0.72	0.93	0.81
WashPost	0.57	1.0	0.73
Weather	0.93	0.93	0.93
Average	0.81	0.81	0.81

Table A.13 – Train on Fossil-Nexus. Test on Huawei-Pixel.

Label	Precision	Recall	F1-score
Bring	1.0	1.0	1.0
Calm	1.0	0.97	0.98
ChinaDaily	0.92	0.79	0.85
Citymapper	1.0	0.93	0.96
DCLMRadio	0.93	0.86	0.89
DiabetesM	1.0	1.0	1.0
Endomondo	0.83	0.86	0.84
FITIVPlus	0.87	0.96	0.92
FindMyPhone	0.67	0.07	0.13
Fit	0.21	0.14	0.17
FitBreathe	0.3	0.57	0.39
FitWorkout	0.5	0.04	0.07
Foursquare	0.93	0.96	0.95
Glide	1.0	1.0	1.0
KeepNotes	0.93	0.96	0.95
Krone	0.96	0.93	0.95
Lifesum	0.96	0.96	0.96
MapMyRun	0.88	1.0	0.93
Maps	0.92	0.82	0.87
Meduza	1.0	1.0	1.0
Mobills	0.74	0.97	0.84
Outlook	0.85	1.0	0.92
PlayStore	0.95	0.71	0.82
Running	0.86	0.89	0.88
SalatTime	0.96	0.96	0.96
Shazam	0.9	0.93	0.91
SleepTracking	0.78	1.0	0.88
SmokingLog	0.97	0.97	0.97
Spotify	1.0	1.0	1.0
Strava	0.81	0.93	0.87
Telegram	0.76	1.0	0.86
Translate	0.8	0.97	0.88
WashPost	0.93	0.96	0.95
Weather	0.96	0.96	0.96
Average	0.86	0.86	0.86

B. Appendix: PriFi (§4.4)

B.1 Table of Symbols

Let λ be a standard security parameter, and let $\mathbb{G}$ be a cyclic finite group of prime order where the Decisional Diffie-Hellman (DDH) assumption [35] holds (*e.g.*, an elliptic curve).

Let $(\texttt{KeyGen}, \mathcal{S}, \mathcal{V})$ be a signature scheme, with $\texttt{KeyGen}(\mathbb{G}, 1^\lambda)$ an algorithm that generates the private-public key pair (p, P) usable for signing. We denote as $\mathcal{S}_p(m)$ the signature of the message m with the key p.

Let $\texttt{KDF} : \mathbb{G}(1^\lambda) \rightarrow \{0,1\}^\lambda$ be a key derivation function that converts a group element into a bit string that can be used as a symmetric key. Let $(\mathcal{E}, \mathcal{D})$ be a ind$-cca2-secure symmetric nonce-based encryption scheme [183]. We denote as $\mathcal{E}_k(m)$ the encryption of the message m with the key k.

Let $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a standard cryptographic hash function. We use the random oracle model to model the output of H. Let $\texttt{PRG} : \{0,1\}^\lambda \rightarrow \{0,1\}^*$ be a standard pseudo-random generator. Let $F_1 : \{0,1\}^\lambda \rightarrow \mathbb{G}$ be a public, invertible mapping function from binary strings to $\mathbb{G}$, and let $F_2 : \{0,1\}^* \rightarrow \mathbb{G}$ be a hash function to $\mathbb{G}$ which maps to any point in $\mathbb{G}$ with uniform probability (*e.g.*, Elligator Squared [211]). Finally, let $F_3 : \{0,1\}^* \rightarrow \mathbb{N}$ be a public function that maps bitstrings to non-zero integers.

Identities. Each party has a long-term key pair (denoted with the *hat* symbol) generated with $\texttt{KeyGen}(\mathbb{G}, 1^\lambda)$:

- $(\hat{p}_i, \hat{P}_i)$ for client C_i, with $i \in \{1, \dots n\}$
- $(\hat{p}_j, \hat{P}_j)$ for guard S_j, with $i \in \{1, \dots m\}$
- $(\hat{p}_r, \hat{P}_r)$ for the relay

Table B.1 – Table of Symbols for PriFi (Section 4.4)

Entities	
C_i	client i.
S_j	guard j.
A	the adversary.
$\mathcal{O}$	the adversary's observations.
Public Parameters	
n	number of clients.
m	number of guards.
b_k	relay's buffer for round k.
Keys	
(p, P)	a (private, Public) key usable for signing.
$(\hat{p}, \hat{P})$	a long-term key pair.
$(\tilde{p}, \tilde{P})$	a shuffled key pair (output of the Neff shuffle in `Setup` (Algorithm 4.1))
$G = (\hat{P}_i, \ldots, \hat{P}_j, \ldots, \hat{P}_r)$	the group definition for an epoch.
$T = (\hat{P}_1, \hat{P}_2, \ldots)$	the long-term roster of allowed public keys for the clients.
Security	
λ	the standard security parameter.
$\mathbb{G}$	a cyclic finite group of prime order (*e.g.,* an elliptic curve).
P	a point in $\mathbb{G}$.
$H : \{0,1\}^* \to \{0,1\}^\lambda$	a cryptographic hash function.
$\texttt{KDF} : \mathbb{G}(1^\lambda) \to \{0,1\}^\lambda$	a key derivation function.
$\texttt{PRG} : \{0,1\}^\lambda \to \{0,1\}^*$	a pseudo-random generator.
$F_1 : \{0,1\}^\lambda \to \mathbb{G}$	a public, invertible mapping.
$F_2 : \{0,1\}^* \to \mathbb{G}$	a hash function that maps uniformly to $\mathbb{G}$.
$F_3 : \{0,1\}^* \to \mathbb{N}^+$	a public mapping from bitstrings to non-zero integers.
$(\texttt{KeyGen}, \mathcal{S}, \mathcal{V})$	a signature scheme.
$(\mathcal{E}, \mathcal{D})$	a symmetric encryption scheme.
DC-net	
$\pi = (\tilde{P}_{\alpha(1)}, \ldots, \tilde{P}_{\alpha(n)})$	the shuffled public keys.
α	the permutation of the shuffle π. The final position of key at index i is $\alpha(i)$.
r_{ij}	the secret shared between client i and server j
m_k	the (intended) upstream message for round k.
m'_k	the upstream message for round k. In the absence of disruption, $m_k = m'_k$.
d	a downstream message.
c_i, γ_i	contributions from client i.
s_j, σ_j	contributions from server j.
x_i	input to DC-net function. $x_i = 0$, or $x_i = m_i$, etc.
z	denotes a bit position. m_z is the z^{th} bit of message m
h_i	history of client i.
$b_{\text{echo\_last}}$	the flag that instructs the relay to echo the last upstream message.
Q_i	contribution from client i in `Blame-Final`
l, l'	upstream, downstream message sizes.

B.2 Protocols

Algorithm B.1: `Anonymize-With-Integrity`
The differences with `Anonymize` (Protocol 4.2) are highlighted in blue.

Inputs: r_{ij}, π, ℓ, ℓ'

Outputs: per round k: m_k, d.

For round $k \in \{1, \dots n\}$:

- Each client C_i sends to the relay $c_i \leftarrow \texttt{DCNet-Gen}(r_{ij}, x_i)$ with

$$x_i = \begin{cases} 0, & \text{if } \alpha(i) \neq k, \\ \text{PoK}_{k',z}(\tilde{p}_{k'} : \tilde{p}_{k'} = \log \tilde{P}_{k'}), & \text{if slot } k' \text{ was disrupted,} \\ m_i || b_{\text{echo\_last}}, & \text{otherwise.} \end{cases}$$

- Each server S_j sends to the relay $s_j \leftarrow \texttt{DCNet-Gen}(r_{ij}, 0)$.
- The relay computes $m_k \leftarrow \texttt{DCNet-Reveal}(c_i, s_j)$
- The relay handles m_k as follows:
  - if m_k is a Blame message, starts the $\texttt{Blame}(\text{PoK}_{k',z}(\tilde{p}_{k'} : \tilde{p}_{k'} = \log \tilde{P}_{k'}), c_i, s_j)$ protocol,
  - else, sends m_k in the appropriate socket in b_k.
- The relay computes and sends to each client $d \leftarrow \text{Downstream2}(b, m_k, k, b_{\text{echo\_last}})$

Function $\texttt{Downstream2}(b, m_k, k, b_{echo\_last})$**:**

$d \leftarrow$ array();

for $k' \in \{1, \dots n\}$ **do**

1. if $k' = k$: // for the anonymous sender:
   (a) add $H(m_k)$ to d, // hash of upstream message
   (b) if $b_{\text{echo\_last}} = 1$: add $\mathcal{E}_{\tilde{P}_k}(m_{k-n})$ to d. // echo last upstream message
2. for each socket $\in b_{k'}$ containing downstream bytes d, add $\mathcal{E}_{\tilde{P}_{k'}}(d)$ to d.

end

Remark. In Algorithm B.2, due to the "Or" format of the PoK, we note that a malicious client can jam their own slot, then send arbitrary Q_i values in step 7 of `Blame`, and still pass the PoK because of their knowledge of $\tilde{p}_k$. However, this has no benefit for the adversary, as mismatching Q_i's are simply checked in the following steps of `Blame`, with no effect on honest parties (Property B.3.5).

Algorithm B.2: `Blame-Final`
The differences with `Blame` (Algorithm 4.3) are highlighted in blue.

Inputs: $\text{PoK}_{k,z}(\tilde{p}_k : \tilde{p}_k = \log\ \tilde{P}_k), c_i, s_j$

1. The relay broadcasts $\text{PoK}_{k,z}(\tilde{p}_k : \tilde{p}_k = \log\ \tilde{P}_k)$ to every client/guard
2. Each client/guard checks the PoK and sends to the relay $\texttt{PRG}(r_{ij})_z, \mathcal{S}_{\hat{p}_i}(\texttt{PRG}(r_{ij})_z)$ for slot k, $\forall i \in \{1,\dots n\}, j \in \{1,\dots m\}$.
3. For each client/guard, the relay checks the signature, and that $\bigoplus_j \texttt{PRG}(r_{ij})_z$ indeed equals $(c_i)_z$ sent in slot k.
4. For each pair of client-guard (C_i, S_j), the relay compares $\texttt{PRG}(r_{ij})_z$ from the client and $\texttt{PRG}(r_{ij})_z$ from the guard. If there is a mismatch, the relay forwards the signed message $\texttt{PRG}(r_{ij})_z$ from C_i to S_j and vice-versa.
5. C_i verifies the signature and checks that the bit $\texttt{PRG}(r_{ij})_z$ mismatches with its own bit $\texttt{PRG}(r_{ij})_z$. Then, he answers with r_{ij} along with a proof of correctness. S_j proceeds similarly.
6. The relay checks the proofs, then uses r_{ij} to recompute the correct value for $\texttt{PRG}(r_{ij})$ for slot k. If no disruptor is identified, the relay continues.
7. Each client C_i computes m values $Q_i = P \cdot F_3(H(\texttt{PRG}(r_{ij}))), \forall j \in \{1,\dots m\}$, where P is the base point of $\mathbb{G}$. Then, each client computes a PoK as follows:

$$\text{PoK}\{\tilde{p}_k, q, q_1 \dots q_m : \tilde{p}_k = \log\ \tilde{P}_k \vee \{ q = \log\ \kappa_i \wedge q_1 := \log\ Q_1 \wedge \vdots \; q_m := \log\ Q_m \wedge q := q_1 + \dots + q_m \}\}$$

   Each client sends a message $Q_i, \text{PoK}, \mathcal{S}_{\hat{p}_i}(Q_i, \text{PoK})$ to R. A non-complying entity is identified as the disruptor. Each server performs similarly, minus the first clause of the PoK which is never true.
8. The relay checks all signatures and PoKs, and potentially identifies a disruptor. If not, for each pair of client-guard (C_i, S_j), the relay compares the two values Q_j, Q_i — they should be equal. If there is a mismatch, at least one of them is lying.
   The relay continues by sending the signed message $Q_i, \text{PoK}, \mathcal{S}_{\hat{p}_i}(Q_i, \text{PoK})$ from C_i to S_j and vice-versa.
9. C_i checks the signature and the PoK, and that a value Q_i is in contradiction with some value of its own. Then, he answers with r_{ij}, the secret shared with S_j, along with a proof of correctness for computing r_{ij}. S_j proceeds similarly. A non-complying entity is identified as the disruptor.
10. The relay checks the proofs of correctness, then uses r_{ij} to recompute the correct values for $\texttt{PRG}(r_{ij})$ and $H(\texttt{PRG}(r_{ij}))$ for slot k, identifying the disruptor.

Once the disruptor is identified, the relay excludes it from the group G and the roster T, and then broadcasts all inputs and messages exchanged in `Blame-Final` to all clients for accountability.

B.3 Security Analysis

Property B.3.1. Let C_0, C_1 be two honest clients who ran `Setup` (Algorithm 4.1) without aborting, and $\alpha(0), \alpha(1)$ the position their respective shuffled key in π. Let $b \in \{0,1\}$ be the value for which the mappings client $\rightarrow$ position $(b \rightarrow \alpha(0), (1-b) \rightarrow \alpha(1))$ are in π. Then, the adversary A cannot guess b with significant advantage.

Proof. This property stems from the direct application of the Verifiable Shuffle used [159].

If `Setup` terminates without aborting for an honest client C_i, then, as a consequence of the checks done by C_i, it means that: (1) the verifiable shuffle π completed correctly, (2) π is signed by every guard in G, including the honest guard, (3) there are at least $K = 2$ clients in π, and (4) their own pseudonym is included in π.

Since both C_1 and C_2 ran `Setup` without aborting, both their shuffled public keys $\tilde{P}_1, \tilde{P}_2$ are included in π. Since both are honest, the corresponding private keys p_1, p_2 are unknown to A. Let $\alpha(0), \alpha(1)$ be the position of C_1 and C_2's respective shuffled keys in π. Due to the hardness of the discrete logarithm in $\mathbb{G}$, A is unable to differentiate between $\tilde{P}_{\alpha(0)}$ and $\tilde{P}_{\alpha(1)}$, and can do no better than random guessing. □

Property B.3.2. Consider one execution of `Anonymize-Final` (Algorithm 4.4). Let $m_1, \ldots, m_k$ be the k output messages. Let C_{i_1}, C_{i_2} be two honest clients, let $k_1 = \alpha(i_1), k_2 = \alpha(i_2)$ be the time slots in which they communicated, and let m_{k_1}, m_{k_2} be the anonymous upstream messages for those slots. Then, A has negligible advantage in guessing $b \in \{1,2\}$ such that m_{k_b} is the message sent by i_1.

Proof. A tries to distinguish between $(C_{i_1} \rightarrow m_{k_1}, C_{i_2} \rightarrow m_{k_2})$ and $(C_{i_1} \rightarrow m_{k_2}, C_{i_2} \rightarrow m_{k_1})$. For simplicity, we define the following equivalent game: for the slot $k \in \{k_1, k_2\}$, A guesses $b \in \{1,2\}$ such that C_{i_b} is the anonymous sender of the message m_k.

To ease notation, let C_{i_1} be C_1 , and C_{i_2} be C_2. Without loss of generality, let S_1 be an honest guard, and let C_1 be the anonymous sender of m_1 during slot k_1. The proof for C_2 as the anonymous sender is analogous.

On the slot k_1, the two clients C_1 and C_2 send the following values:

$$\begin{cases} c_1 & \leftarrow \bigoplus_{j=1}^{m} \texttt{PRG}(r_{1,j}) \oplus x_1 \\ \kappa_1 & \leftarrow F_1(\gamma) + F_2(h_1) \cdot \sum_{j=1}^{m} F_3(H(\texttt{PRG}(r_{1,j}))) \end{cases} \tag{B.1}$$

$$\begin{cases} c_2 & \leftarrow \bigoplus_{j=1}^{m} \texttt{PRG}(r_{2,j}) \\ \kappa_2 & \leftarrow F_2(h_2) \cdot \sum_{j=1}^{m} F_3(H(\texttt{PRG}(r_{2,j}))) \end{cases} \tag{B.2}$$

... with $x_1 \leftarrow \mathcal{E}_\gamma(m_1) \,||\, b_{\text{echo\_last}}$.

Since S_1, C_1 and C_2 are honest, the values $\texttt{PRG}(r_{1,1})$ and $\texttt{PRG}(r_{2,1})$ are unknown to A. We isolate the contribution of S_1:

$$\begin{cases} c_1 & \leftarrow \bigoplus_{j=2}^{m} \texttt{PRG}(r_{1,j}) \oplus \texttt{PRG}(r_{1,1}) \oplus x_1 \\ \kappa_1 & \leftarrow F_1(\gamma) + F_2(h_1) \cdot F_3(H(\texttt{PRG}(r_{1,1}))) + F_2(h_1) \cdot \sum_{j=2}^{m} F_3(H(\texttt{PRG}(r_{1,j}))) \end{cases} \tag{B.3}$$

$$\begin{cases} c_2 & \leftarrow \bigoplus_{j=2}^{m} \texttt{PRG}(r_{2,j}) \oplus \texttt{PRG}(r_{2,1}) \\ \kappa_2 & F_2(h_2) \cdot F_3(H(\texttt{PRG}(r_{2,1}))) + F_2(h_1) \cdot \sum_{j=2}^{m} F_3(H(\texttt{PRG}(r_{2,j}))) \end{cases} \tag{B.4}$$

We rewrite the terms known to the adversary as $\text{cst}_1, \ldots, \text{cst}_4$:

$$\begin{cases} c_1 & \leftarrow \text{cst}_1 \oplus \texttt{PRG}(r_{1,1}) \oplus x_1 \\ \kappa_1 & \leftarrow F_1(\gamma) + F_2(h_1) \cdot F_3(H(\texttt{PRG}(r_{1,1}))) + \text{cst}_2 \end{cases} \tag{B.5}$$

$$\begin{cases} c_2 & \leftarrow \text{cst}_3 \oplus \texttt{PRG}(r_{2,1}) \\ \kappa_2 & \leftarrow F_2(h_2) \cdot F_3(H(\texttt{PRG}(r_{2,1}))) + \text{cst}_4 \end{cases} \tag{B.6}$$

The adversary must output $b \in \{1,2\}$; he wins if C_b is the anonymous sender on slot k_1, or equivalently, if $\alpha(i_b) = k_1$. The observations of the adversary are $\mathcal{O} = (\pi, (c_1, \kappa_1), (c_2, \kappa_2))$. We analyze the observations component by component:

- If `Setup` was done correctly, the Neff shuffle π (with permutation α) does not give A any advantage in guessing b (Property B.3.1).
- Both c_1, c_2 have the same length, and both $\texttt{PRG}(r_{1,1})$ and $\texttt{PRG}(r_{2,1})$ are indistinguishable from random bits to A. Thus, c_1 and c_2 are indistinguishable from each other.
- Both terms $F_3(H(\texttt{PRG}(r_{1,1})))$ and $F_3(H(\texttt{PRG}(r_{2,1})))$ are non-zero integers due to F_3. They are unknown to A. Both terms $F_2(h_1), F_2(h_2)$ are non-zero points in $\mathbb{G}$. Finally, $F_1(\gamma)$ is a point uniformly distributed in $\mathbb{G}$. Due to the hardness of the logarithm in $\mathbb{G}$ (which is implied by the hardness of the Decisional Diffie-Hellman problem), the adversary cannot distinguish $P_\gamma + a \cdot P_1$ from $b \cdot P_2$, with a, b, P_γ unknown.

Hence, with C_1 as the anonymous sender, the adversary cannot distinguish the pairs $(c_1, \kappa_1), (c_2, \kappa_2)$: it can only guess b with probability $1/2$ and advantage 0. As desired, we have that the observations $\mathcal{O}$ of the adversary are indistinguishable whether a particular honest user is the sender or not. □

Property B.3.3. The anonymity of any honest client is unaffected by the information revealed to A during the execution of `Blame-Final`: $\texttt{PRG}(r_{ij})_l$ in step 2, r_{ij} made public in step 5 and 9, and finally, Q_i and PoK sent in step 7 of `Blame-Final` (Algorithm B.2).

Proof. The proof is performed in four steps:

1. Revealing the $z-$th bit of the values $\texttt{PRG}(r_{ij})$ does not affect the anonymity of honest clients: all sent a 0 in this position.
2. Revealing the shared secret r_{ij} between a client C_i and a server S_j only happens if either C_i or S_j is malicious, in which case r_{ij} is already known to A.
3. The values Q_i do not reveal any information except the correctness of $F_3(H(\texttt{PRG}(r_{i,j})))$ due to the hardness of the discrete logarithm in $\mathbb{G}$.
4. Trivially, the zero-knowledge PoK reveals no information.

□

Proof of point 1. The values $\texttt{PRG}(r_{ij})_z$ for slot k de-anonymize precisely the z^{th}-bit of the slot k, by revealing the composition of the messages c_i and s_j at position z.

Let C_i be the honest owner of slot k. Only C_i can generate the proof of knowledge $\text{PoK}_{k,z}(\tilde{p}_k : \tilde{p}_k = \log \tilde{P}_k)$, and honest clients verify the PoK before revealing any information, hence honest clients do not reveal information about a slot without the owner's consent.

In slot k, all non-sender, honest clients transmitted a 0 on bit z, since $k \neq \alpha(i)$ and the protocol forces them to transmit 0. Additionally, the slot owner C_i only sends PoK for z such that $(m_k)_z = 0$. Therefore, on slot k, at position z, all honest clients transmitted a 0. The values revealed match the $(c_i)_z$ values previously sent, and the $(c_i)_z$ are composed solely of the output of the PRGs seeded by the secrets, which are indistinguishable from a random string; the knowledge of one given bit reveals nothing about other parts of the PRG output. □

Proof of point 2. By revealing r_{ij}, a pair of client-guard (C_i, S_j) completely reveal their contributions to the DC-net. However, an honest client and an honest guard never contradict each other, as the protocol ensures that their values $\texttt{PRG}(r_{ij})_l$ are correctly computed from the common shared secret r_{ij}. Since in step 5, an honest client C_i only reveals r_{ij} when seeing (1) a signed message from a guard, (2) for which there is a contradiction, he never reveals r_{ij} for an honest guard S_j. The same argument can be made for S_j about any honest client.

Therefore, those values are revealed only when C_i or S_j is malicious (or both); in this case, A already had knowledge of r_{ij}. □

Proof of point 3 and 4. The Q_i are independent from the communicated content and the slot owner. Both senders and non-senders compute Q_i in the same way. Due to the hardness of the discrete logarithm problem in $\mathbb{G}$, A cannot recover $H(\texttt{PRG}(r_{ij}))$ from $Q_i = P \cdot F_2(H(\texttt{PRG}(r_{ij})))$. Finally, the zero-knowledge proof of knowledge PoK trivially reveals no information. □

Property B.3.4. If $\exists i, j$ two honest clients who received $d_i \neq d_j$ on round k, then nor the relay, nor A can decrypt m_k for any subsequent rounds $k' > k$.

Proof. Without loss of generality, let C_1, C_2 be two honest clients who received $d_1 \neq d_2$ at the end of round k. Then, with overwhelming probability, $h_1 \neq h_2$ due to the use of the cryptographic hash function H, and it follows that $F_2(h_1) \neq F_2(h_2)$ with high probability.

The message m_k is encrypted with γ using $\mathcal{E}$, an ind\$-cca2-secure symmetric encryption scheme. The honest sender transmits γ blinded as follows:

$$\kappa_1 = F_1(\gamma) + F_2(h_1) \cdot \sum_{j=1}^{m} F_3(H(\texttt{PRG}(r_{1,j})))$$

Consider a unique honest guard S_1. As done in the previous proofs, we isolate the contribution from honest parties and replace the remainder with a constant:

$$\kappa_1 = F_1(\gamma) + F_2(h_1) \cdot F_3(H(\texttt{PRG}(r_{1,1}))) + \text{cst}_1$$

Since he is not the sender, C_2 sends $\kappa_2 = F_2(h_2) \cdot F_3(H(\texttt{PRG}(r_{1,1}))) + \text{cst}_2$.

The guard S sends $\sigma_1 = -\sum_{i=1,2} F_3(H(\texttt{PRG}(r_{i,1}))) + \text{cst}_3$.

In the absence of equivocation, the relay is able to compute $F_1(\gamma) \leftarrow \kappa_1 + \kappa_2 + F_2(h_r)\sigma_1 + \text{cst}$, with $h_r = h_1$. However, the values κ_1 and κ_2 have been blinded with a different factor $F_2(h_1)$

and $F_2(h_2)$.

$$\begin{aligned}\gamma' = \gamma &+ (F_2(h_r) - F_2(h_1)) \cdot F_3(H(\texttt{PRG}(r_{1,1}))) \\ &+ (F_2(h_r) - F_2(h_2)) \cdot F_3(H(\texttt{PRG}(r_{2,1}))) \\ &+ \text{cst}\end{aligned}$$

If A sets $h_r = h_1$, then $\gamma' = \gamma + (F_2(h_2) - F_2(h_1)) \cdot F_3(H(\texttt{PRG}(r_{2,1}))) + \text{cst}$.
If A sets $h_r = h_2$, then $\gamma' = \gamma + (F_2(h_1) - F_2(h_2)) \cdot F_3(H(\texttt{PRG}(r_{1,1}))) + \text{cst}$.

In both cases, $\gamma' \neq \gamma$ and the adversary cannot recover γ.

□

Property B.3.5. An honest relay is never accused of an equivocation attack.

Proof. Since the downstream messages d are signed by the honest relay, the adversary is unable to forge a message sent by the honest relay. Since an honest relay never equivocates, a malicious client attempting to frame the relay is equivalent to a malicious client sending wrong c_i, κ_i values.

When this happens during an honest client slot, it will eventually start a `Blame` procedure. The `Blame` procedure verifies the correctness of the c_i, κ_i values since, for a correct h_i, the c_i, κ_i of all parties but the honest sender are fully determined by the shared secrets r_{ij}. The honest sender trivially sends the correct values.

When this happens during the slot of a malicious client, the client may wrongly compute κ_i and still pass the `Blame` due to the first "Or" of the PoK. In this case, no party is identified as a disruptor.

□

Property B.3.6. Let C_i be the owner of a slot k, and let C_d, $d \neq i$ be another client (or guard). If C_d sends an arbitrary value c'_i instead of the value c_i (or s_j) as specified in the protocol, and if `Blame-Final` is started by the relay, then, one malicious client (or guard) is identified as the disruptor and is excluded from subsequent communications.

Proof. The proof assumes a disrupting client; the argument for a disrupting guard is analogous.

Let c'_i be the value sent instead of c_i. Then, we can write $c'_i = c_i \oplus q$ for some value q, and we observe that $m_k = m_i \oplus q$. Let u be the number of 0 bits in m'_i. The adversary disrupts stealthily (*i.e.,* is not detected) if no 0 in m_i is flipped to a 1: in this case, the honest client does not start `Blame-Final`.

Since $m'_i = \mathcal{E}_\gamma(m_i)$ and $\mathcal{E}$ is ind\$-cca2-secure, m_i is uniformly distributed in $\{0,1\}^{|m_i|}$. The probability that no 0-bit is flipped follows $(\frac{1}{2})^u$, with $\mathbb{E}[u] = \frac{|m_i|}{2}$. Thus, a malicious client is excluded with probability exponentially decreasing in the message length $|m_i|$. □

Property B.3.7. If a client C_i sends an arbitrary value κ'_i instead of the value κ_i as specified in the protocol, then, with acceptable probability, C_i is identified as the disruptor and is excluded from subsequent communications.

Proof. The argument is analogous to the proof of Property B.3.6: if the honest client whose slot was disrupted can start a blame, then the `Blame` protocol ultimately excludes one disruptor. □

Property B.3.8. An honest client is never excluded as a disruptor.

Proof. The end of `Blame` (Algorithm 4.3) describes more precisely what it means for a client C_i to be "excluded as a disruptor": the relay broadcasts a new group G' and roster T' without the public key of C_i. The process for a server is analogous, except that $T' = T$.

The proof is done for an honest client, and is analogous for an honest server.

Trivially, an honest client always produces correct values. During a `Blame` procedure, every message is signed. The honest client first sends $\texttt{PRG}(r_{ij})$ for slot k, which matches what he sends in round k. Then, if he is not in contradiction with some server, the honest client stops taking part in `Blame` (and is not excluded); let us assume he is in contradiction with S_j. Then, upon reception of the signed message from S_j, he reveals r_{ij} along with a proof of correctness.

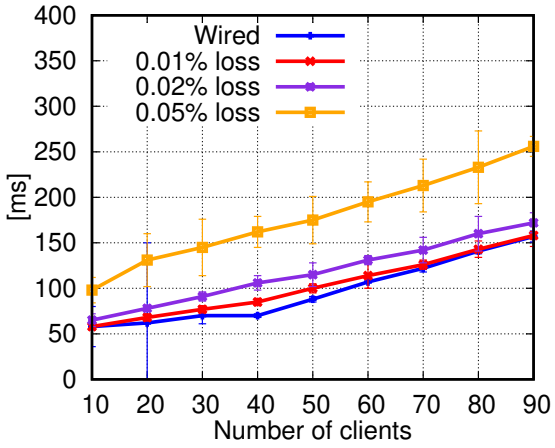
To exclude a client (by broadcasting a new group G' and roster T), the relay also have to reveal all inputs of `Blame` for accountability. To exclude an honest client without blatantly cheating, the relay would have to forge one signature, which contradicts our adversarial model (Section 4.4.3.2).

We note that in practice the malicious relay could update G and T without justification; however, this is analogous to a simple denial-of-service and contradicts the system model (Section 4.4.3.1). In practice, this would prompt clients to take administrative actions against the relay. Finally, even when that is the case, the honest client has not been "framed as a disruptor". □

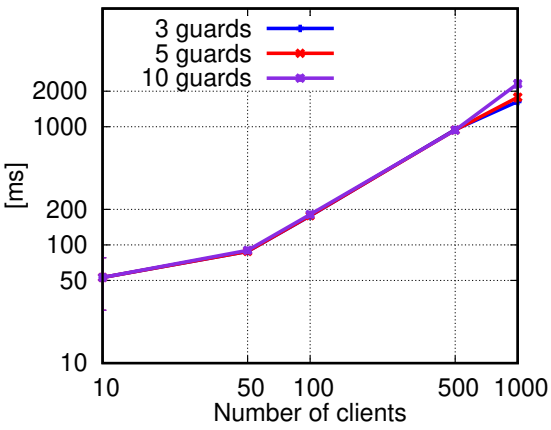
Property B.3.9. An honest entity is never identified as a disruptor.

Proof. Trivially, honest clients follow the protocol. Each step of the `Blame` protocol consists of revealing some already-performed step of `Setup` and `Anonymize` protocol; by definition, honest entities performed these steps correctly. □

B.4 Scalability



(a) Latencies when varying the loss rate.



(b) Larger-scale scalability. This experiment's purpose is to understand the limits of the system.

Loss Rates. We briefly explore the impact of various loss rates in Figure B.1a. While an imperfect representation, this experiment could sketch the performance in a real WLAN. The results show that the current implementation requires "good" link quality (loss rates ≤ 10%, see [195] Figure 3a) to maintain low latency, that then degrades rather quickly with increasing loss rates. We note that current WLANs typically have good resilience to message drops; noise and collisions result in increased jitter rather than losses [131]. Implementing PriFi directly on Network Interface Cards (NIC) could give better control and performance. Finally, we note that WLANs have a less expensive broadcast than LANs, a factor not reflected in this experiment.

Scalability. Figure B.1b shows the performance for larger anonymity sets and with more guards. While the number of guards has almost no impact on performance due to the buffering at the relay, our implementation would not be low-latency for more than a few hundred clients.

B.5 Different Scenarios

Local Trusted Guard. The ICRC benefits from the particular situation of *Privileges and Immunities* (P&I) [72], legal agreements with governments that provide a layer of defense in order to operate in environments of armed conflicts and other situations of violence. In practice, P&I notably grants the delegations with inviolability of premises and assets. Together with the strong physical security deployed at their server rooms, each delegation essentially has a local trusted server. Aside from the ICRC, P&I can apply to embassies and diplomatic missions [217].

We simulate this new deployment with one guard in the LAN instead of three remote guards. The latency between the relay and the unique guard is 10 ms. In this case, we observe that the latency experienced by clients is roughly cut in half, shown in Figure B.1, purple dotted curve versus blue solid curve. An additional benefit is that the only WAN bandwidth usage is the anonymized goodput.

VPN. When a member of an organization is accessing the network remotely (*e.g.,* when traveling), it can benefit from PriFi's protection from outside the organizational LAN. The cost is in performance, as the relay waits upon the slowest client to decode an anonymous message.

We simulate this alternative deployment scenario by having *all* clients outside the LAN. This is modeled by setting the latency between clients and relay to 100 ms instead of 10 ms. In this case, the baseline for latency is 200 ms. We observe that in this scenario, end-to-end latency varies from 280 ms to 498 ms as shown in Figure B.1, red solid curve versus blue solid curve. While this latency is too high to support VoIP and videoconferencing, it might be acceptable for web browsing. We note that all users are slowed down. Although not explored in this work, this slowdown could be mitigated by having two PriFi networks, one reserved for local users, the other accepting remote users. Local users would participate in both networks, ensuring a sufficiently large anonymity set, and would communicate only using the fastest PriFi network.

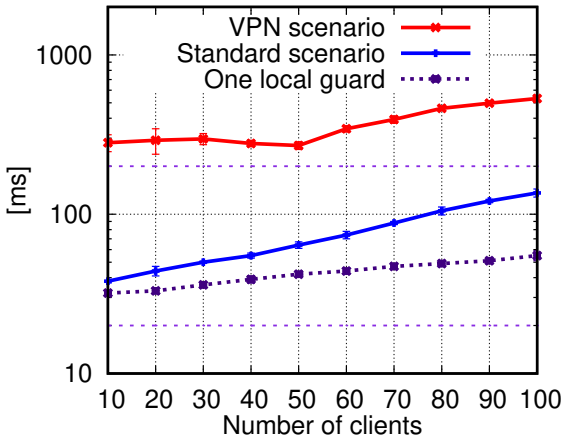


Figure B.1 – Latencies observed in the standard scenario (3 remote guards with high-latency towards the relay), when having one trusted guard in the LAN, and in the VPN scenario (all users are in the WAN setting, with 100 ms latency to the relay). The baseline for the VPN scenario is 200 ms, and 20 ms in the two other scenarios.

B.6 Client Churn

In DC-nets, churn invalidates the current communications and leads to data re-transmissions and global downtime where no one can communicate. Although re-transmissions are acceptable with PriFi's small and frequent rounds (*e.g.,* a few 100KB of payload each 10ms), frequent churn could prevent delay-sensitive applications from running on top of PriFi. To our knowledge, our contribution here is the first analysis of the impact of churn on DC-nets in a realistic scenario where nodes are mobile (*e.g.,* wireless devices).

Dataset. To characterize node mobility, we use a standard dataset [170] from CRAWDAD. The dataset contains 4 hours of traffic recorded in a cafeteria, and includes the devices' association and disassociation requests. It has 254 occurrences of churn consisting of 222 associations (33 unique devices) and 32 disconnections (12 unique devices).

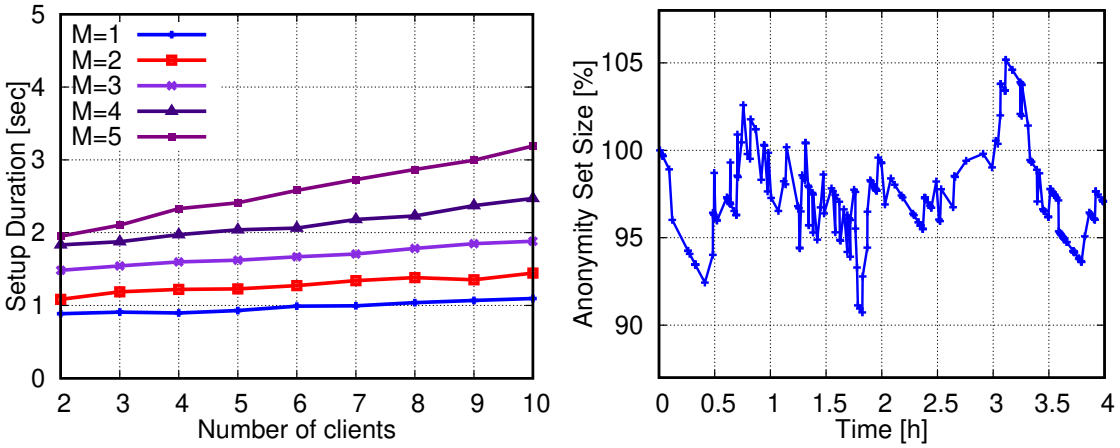
Dataset Analysis. Each device (dis)connection induces a re-synchronization time of D milliseconds (for `Setup`), where D is dominated by the number of guards m and clients n and the latency between them. A typical value for D would be a few seconds (Figure B.1a). We analyze two strategies to handle churn:

We analyze two strategies to handle churn:

- The *naïve* approach stops communication for D seconds unconditionally at every churn event.
- An *abrupt* disconnection stops communication for D seconds at each disconnection only. Devices connect using a graceful approach where `Setup` is done in the background, keeping the previous `Anonymize` protocol running until the new set of clients is ready. This strategy can be enforced by the relay.

Table B.2 – Effect of different churn handling strategies on PriFi.

Strategy	**# Interrupt.**	**Availability [%]**	**Max Downtime [s]**
Naïve	254	98.72792	1.55147
Abrupt	32	99.81778	0.82



(a) Experimental evaluation of the network downtime (*i.e.,* where no communication can happen) in case of abrupt disconnection, averaged 10 times, with 95% percentiles. The time is lower-bounded by the latency to the m guards.

(b) Size of the anonymity set versus time, when using PriFi in the café scenario. This shows among how many users a PriFi client is anonymous.

Figure B.1 – Pipelining and ICRC.

In Table B.2, we show (1) the raw number of communication interruptions, which directly comes from the node mobility in the dataset, (2) the network availability percentage, computed as $\frac{1-\text{downtime}}{\text{total time}}$, and (3) the maximum continuous downtime, *i.e.,* the longest network interruption.

Analysis. Using PriFi in the cafeteria scenario represented by the dataset [170] would naturally decrease network availability, as churn induces global downtime. Over 4h, between 0 and 32 global losses of communication occur, and the network availability ranges between 99.82% and 100%, depending on the disconnection types. The longest disconnection period is 0.82s in the worst case, which might be noticeable as a slight lag by users of time-sensitive applications (*e.g.,* VoIP). If we extrapolate the worst-case availability 99.82% over 10 minutes, the total expected downtime is below 1.1 second. We note that the metric does not directly compare with the classical "five 9's" in Service Level Agreements, for which in the worst case all downtime can be continuous (*e.g.,* a server is down for a few minutes); in our model, downtime is likely to be spread over many short events over time, following user mobility.

Anonymity Metrics. We now analyze the size of the anonymity set with respect to time, *i.e.,* among how many participants a PriFi user is anonymous at any point in time (Figure B.1b). In particular, we are interested in the variations, which are due to user mobility in this case.

We display the difference, in percentage, between the actual anonymity set size and the baseline tendency. We see that size of the anonymity set does not vary more than ±8%, and the mean number of users is 50. Hence, our estimation for the worst-case of "anonymity loss" in this scenario is of 4 users, which seems acceptable in an anonymity set of 50 users.

C. Appendix: Rubato (§4.5)

C.1 Service provider API

```
type Onion = [MessageSize + SymmetricOnionOverhead]byte

type RoundData struct {
    Onion [2]Onion      // Messages are split into two, one for each buddy circuit
    OutboundMsg []byte  // Previously sent message for this round
}

rpc (t *Txn) GetAddressBook() []byte                                    // RPCs for the Address Book
rpc (t *Txn) SetAddressBook(data []byte)

rpc (t *Txn) SetCircuitSetup(epoch int, onions [MaxBuddies][2][]byte) // Circuit Setup RPCs
rpc (t *Txn) GetCircuitSetupAck(epoch int) [MaxBuddies][2][]byte      // each buddy has 2 circuits

rpc (t *Txn) GetOutgoingMessage(epoch int, round int) RoundData       // RPCs to send messages
rpc (t *Txn) SetOutgoingMessage(epoch int, round int, rd RoundData)

rpc (t *Txn) GetHeaders(epoch int, round int) [MaxBuddies][2]byte     // RPCs to get messages
rpc (t *Txn) ShuffleInbox(ShuffleParams) [][]byte
rpc (t *Txn) FetchInbox(DHkey []byte, indices []int) [][]byte
```

Figure C.1 – Service provider client API. Clients use these RPCs asynchronously to setup circuits to their buddies and send/receive messages through them.

Ludovic Barman

Nationality: Swiss
Date of birth: 17/06/1992
Current location: Lausanne, Switzerland
Languages: Fluent in French/English (C2), B2 in German/Italian
Contact: ludovic.barman@protonmail.com

Core skills: Research in security and privacy. Software development (Web/desktop). Strong general knowledge in networking, security and applied cryptography. Project management. Preparing and delivering talks for different types of audience.
Technologies: Go, Typescript/JS, Python. Decent experience in Scala, Java. Web development (HTML/PHP/SQL). Good knowledge of basic infra, testing & deployment (docker, Travis CI, Jasmine, puppeteer).

The information below is compressed to fit in one page. The extended version is available at https://lbarman.ch

List of publications:

- L. Barman, M. Kol, D. Lazar, Y. Gilad, N. Zeldovich. **Rubato: Metadata-Private Messaging for Mobile Devices**. Under submission
- L. Barman, A.Dumur, A. Pyrgelis, J-P. Hubaux. **Traffic-Analysis of Wearable Devices over Bluetooth Classic and BLE**. Ubicomp 2021
- L. Barman, M. Zamani, I. Dacosta, A. Pyrgelis, B. Ford, J-P. Hubaux, J. Feigenbaum. **PriFi: Low-Latency Metadata Protection for Organizational Networks**. PETS 2020
- C. Troncoso et al. **Decentralized Privacy-Preserving Proximity Tracing**. arXiv 2020
- L. K. Nikitin, L. Barman, M. Underwood, W. Lueks, B. Ford, J-P. Hubaux. **Reducing Metadata Leakage from Encrypted Files and Communication**. PETS 2019
- L. Barman, M. Zamani, I. Dacosta, B. Ford, J-P. Hubaux, J. Feigenbaum. **PriFi: A Low-Latency [...] Protocol for Local-Area Anonymous Communication**. WPES 2016
- F. Armknecht, L. Barman, J-M. Bohli, G. Karame. **Mirror: Enabling Proofs of Data Replication and Retrievability in the Cloud**. USENIX Security 2016
- L. Barman, E. Graini, J-L. Raisaro, E. Ayday, J-P. Hubaux. **Privacy Threats and Practical Solutions for Genetic Risk Tests**. GenoPri 2015

Education:

EPFL, Lausanne:	PhD in Security and Privacy, with prof. Jean-Pierre Hubaux and prof. Bryan Ford	*since 2015*
EPFL, Lausanne:	Master in Communications Systems, specialized in Security	*2013-2015*
NUS, Singapore:	One-year exchange in Electrical Engineering & Computer Science	*2012-2013*
EPFL, Lausanne:	Bachelor in Communications Systems	*2010-2013*

Selected professional experiences:

- **Teaching assistant** at **EPFL** (part of the PhD program) *since 2015*
  In the class "Information Security and Privacy", I rebuilt the aging infrastructure in favor of a more reliable setup with dockers and Continuous Integration. I also designed several exercises such as a TLS downgrade attack & implementation of a PAKE protocol.
  In the class "Mobile Networks", I helped build hands-on exercises about Wireless networks and their security/privacy aspects. I gave a lecture about Tor and the anonymity on the Web.
- **Intern (Master Project)** at **NEC Laboratories Europe**, Heidelberg *2015*
  I implemented and benchmarked a research project in Scala, later published at USENIX Security.
- **Teaching assistant** at **EPFL** *2012*
  For the class "System-oriented Programming", where students learn about SH, C, Perl, Unix.
  Supervision of semester projects for students in Java, involving cryptography and networking.
- Various **Web Developer** positions (Sunergic, CJ Online Works, JE EPFL, Intemporare) *2008-2019*
  At Sunergic, I created a Web application for monitoring Siemens solar panels. In addition, it enabled clients to design a roof (through a graphical wizard) and estimate the expected efficiency and profit of a solar installation. This application has been used by several partners of Sunergic and Romande Energie