UNIVERSITY OF CAMBRIDGE

Strong metadata privacy for mobile devices and applications

Daniel Hugenroth

Darwin College

This dissertation is submitted on 29th September 2023 for the degree of *Doctor of Philosophy*.

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any work that I have submitted, or am concurrently submitting, for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my dissertation has already been submitted, or is being concurrently submitted, for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. This dissertation does not exceed the prescribed limit of 60 000 words.

Daniel Hugenroth
29th September 2023

Abstract

Strong metadata privacy for mobile devices and applications
Daniel Hugenroth

Smartphones have become the primary computing devices for many. Living inconspicuously in our pockets, they store our most intimate personal messages and pictures as well as sensitive corporate information and government secrets. This has already motivated widespread adoption of end-to-end encryption for mobile messaging applications, such as WhatsApp and Signal, which protect the confidentiality of messages. However, metadata, such as who has been messaging whom and when, can still be observed by platform operators, local internet providers, and other adversaries tapping into network traffic. This dissertation presents protocols and applications for mobile devices that not only protect the content of messages but also communication patterns.

Anonymity networks provide metadata privacy, but the most popular ones, like Tor, remain vulnerable to traffic analysis, while strong alternatives, like Loopix, use cover traffic at the expense of higher bandwidth and latency. In this context smartphones raise two important challenges: battery constraints dictate conservative power usage and connectivity is often intermittent.

In order to better understand power consumption on modern smartphones we run experiments on real hardware and find that cryptographic operations are cheap while radio transmission can be costly. In particular, popular solutions such as VPN and Tor are practical with negligible impact on the battery life. However, more secure designs using cover traffic are impractical and highlight the need for protocol design that takes energy limitations into account.

The latency and bandwidth requirements of protocols with strong metadata privacy are particularly challenging when sending messages to many recipients—especially on mobile devices where users are often offline. We design Rollercoaster, a multicast scheme for mix networks which incorporates these constraints and allows better utilisation of the underlying network for sporadic group communication. This enables decentralised applications such as group messaging and collaborative text editing while retaining efficient mix parameters.

Finally, we present CoverDrop, a practical system for initial contact between whistleblowers and journalists. CoverDrop integrates into a standard news reader app such that all its users contribute cover traffic to achieve unobservable communication for sources while having negligible impact on battery life. In addition, we implement plausibly-deniable storage to keep previous usage of CoverDrop secret even if the phone is captured by an adversary. To achieve this, our key stretching scheme, called Sloth, uses the Secure Element found in many modern smartphones, preventing the adversary from parallelising brute-force attacks and therefore allowing for shorter, more memorable passphrases.

Acknowledgements

First, I would like to thank my supervisor Alastair Beresford. I am indebted to him for providing me with immensely helpful guidance throughout my entire PhD and giving me all the support and freedom I could have ever asked for. I would not be where I am today without him and the journey would have been certainly far less enjoyable. This extends to all kind and inspiring people that I met along the way in both industry and academia. In particular to my lab mates and collaborators Alberto, Ceren, Diana, Jenny, Jovan, Luis, Martin, Michael, Nicholas, Sam, and the CoverDrop team.

I thank my family—Ulla, Martin, Christopher, and Marie—for their unwavering trust in my decisions and always encouraging me to pursue my goals. I cannot know what went through your heads when you placed an old i486 DOS machine in my room, but here we are. However, doing a PhD abroad was an unlikely path for me to take. For this, I am thankful to the Bundeswettbewerb Informatik and the Studienstiftung des deutschen Volkes for broadening my horizon and helping me discover my path.

Thank you to my friends from both past and present. A PhD is a marathon and not a sprint which means that one should definitely stop along the way to go punting, rowing, travelling, and simply enjoying each other's company. You made the last four years special and memorable. Thank you, Andi, Anna, Armin, Benedikt, Faustyna, Felix, Frede, Friedrich, Jacob, James, Jannik, Kat, Maline, Martin, Saskia, Simone, Tiana, and everyone who belongs on this list.

And thank you, dear reader, for your interest in this dissertation.

Table of contents

List of figures

Chapter 1

Introduction

Over the last 10 years end-to-end encrypted (E2EE) communication has achieved widespread adoption around the globe. Thanks to Signal, WhatsApp, and other apps, billions of users communicate knowing that the content of their messages can only be read by them and the intended recipients. This confidentiality is critical as messaging apps gained widespread adoption from individuals exchanging private messages to large corporations sharing sensitive business information. Of course, this also includes the highest roles in government as exemplified by Germany's ex-chancellor Angela Merkel who was dubbed "Handykanzlerin" [21] (mobile phone chancellor) due to her pervasive use of text messaging. In previously deployed solutions the encryption, and thus confidentiality, terminated at the servers of the platform provider requiring that the user trusts them to not read or modify their messages. This is why end-to-end encryption is considered such an important improvement.

However, even E2EE systems leak metadata which comprise information such as who communicates with whom and when. A local system administrator or an Internet service provider (ISP) can easily record the senders and recipients of all connections via their network. Similarly, the operators of messaging services can record the senders and recipients of all messages, as this information is included in the headers to correctly forward messages to the recipient devices. And while these operators do not necessarily have a motive to spy their users, they might be (legally) compelled to record such data or their systems might be compromised by attackers. This communication metadata alone, without knowing the content of messages, is already powerful. It could be used to compile a list of people a journalist contacted to then unmask whistleblowers; or to identify a group of activists who worked together by mapping their social network starting from a publicly known leader.

Anonymity networks are privacy enhancing technologies (PETs) which allow users to hide such information and thus guarantee metadata privacy. From its inception in the 1980s through Chaum's seminal works on untraceable communication [32, 34], the research field produced numerous designs that achieve meaningful trade-offs between security and performance. This field is also a very practical one. For instance, Tor [50] is an anonymity network that is used world-wide and deployed across many continents. As such, anonymity networks today protect whistleblowers who reach out to journalists to report wrongdoing; they allow activists and non-governmental organisations (NGOs) to communicate and coordinate in precarious conditions; and they allow individuals to evade local censorship measures. However, this technology can also be abused by criminals to evade detection by law enforcement

and avoiding consequences for their actions. Examples of this are online marketplaces for illicit goods where both parties remain anonymous towards each others. We explore the dual-use nature of anonymity networks and discuss some ethical aspects in Section 2.2.

Tor works by routing all traffic between the user and their communication partner, for instance a web server, via a route of three user-chosen intermediate nodes that are called onion routers. This route is typically established at the beginning of a session, eliminating the need to include detailed routing information in later packets. Onion routers are operated by volunteers and organisations in many different countries so that the traffic typically crosses multiple jurisdictions and therefore making comprehensive surveillance difficult. The individual packets are encrypted in multiple layers such that each node along the route can only decrypt the outer-most layer which then reveals the address of the next hop and the next encrypted payload. Since each onion router only knows their direct predecessor and successor along the route, no node learns the identity of both the initial sender and final recipient.

However, Tor and similar systems are susceptible to traffic analysis attacks by adversaries that can observe parts of its network [45, 95, 145]. In particular, Tor is vulnerable to confirmation attacks where an adversary might start with the suspicion that two particular users are communicating with each other. If the adversary observes that outgoing packets at A correlate with incoming packets at B, they can accept this as confirmation that A is talking to B. The adversary can try to repeat these observations over multiple rounds to build further confidence. The practical relevance of network-level attacks on a global level gained more public attention with the Snowden revelations in 2013 which provide evidence for existing large-scale metadata surveillance [88]. Section 2.1.2 discusses Tor and traffic analysis attacks in more detail.

Other designs can offer stronger protection. A successful archetype that is central to this dissertation are mix networks with cover traffic, such as Loopix [110]. In mix networks each message is independently routed and delayed which makes traffic analysis more difficult. In addition, all clients send outgoing messages following an independent schedule to hide communication patterns. When the client does not have any real messages to be sent, so called cover messages are sent instead. While providing very strong metadata privacy, such systems require trade-offs between efficiency, latency, and performance. The Loopix system allows for low-latency communication and its provider nodes can cache messages when clients are offline. Because of their practicality and strong metadata privacy, I made mix-based architectures, and in particular Loopix, the central anonymity technology in my research. The practical relevance of Loopix is further demonstrated by Nym [49] which is an active real-world implementation that is deployed across hundreds of active mix nodes in many countries. Section 2.1.3 discusses mix-based networks, Loopix, and Nym in more detail.

However, smartphones have become the primary computing devices for most users and this challenges existing anonymity network architectures which were designed with standard desktop computers in mind. In particular, mobile devices have limited battery capacity. The energy measurements in Chapter 3 show that cover traffic, which is critical for strong-metadata privacy, becomes infeasible for parameters that are commonly chosen today. Hence, understanding the power consumption of smartphones and applications is critical when evaluating modern anonymity network designs. In addition, as mobile devices travel with the user, they switch between mobile and WiFi networks and also go offline completely in areas without network coverage. This unreliable connectivity requires

that the anonymity network can cache or retransmit messages when the mobile client is offline. These observations were defining constraints for the development of the Rollercoaster protocol that I present in Chapter 4.

In recent years, smartphones have provided new opportunities to develop more secure and usable privacy enhancing technologies (PETs). The sandboxing of apps increases the confidence of developers that local data is stored confidentially and it reduces the risk of side-channel attacks from other apps. Most smartphones also include biometric authentication using fingerprint readers and face recognition technology which allows for integration of more frequent user authorisation without creating security fatigue from entering passwords. Finally, modern smartphones can guard secret keys in a dedicated Secure Element (SE) which can resist even physical attacks. The Sloth protocol (Chapter 5) leverages the SE to provide plausibly-deniable storage that provides definite time guarantees against brute-force password guessing. An implementation with similarly strong security guarantees for the average desktop computer would not be easily possible. Such building blocks are important for PET applications. In the CoverDrop system (Chapter 6) they enable whistleblowers to convincingly deny their activities in case their smartphone is captured.

In fact, the anonymity network providing an infrastructure layer and the applications running on top of it are typically intertwined. This becomes apparent in the projects that I include in this dissertation. The motivation for Rollercoaster (Chapter 4) arose from building decentralised collaboration software on top of Loopix. For this, updates must be shared with others through multicast operations. However, the intentionally limited sending rate of mix networks renders naïve approaches infeasible, and hence Rollercoaster is a specialised multicast protocol that is tailored to these unique constraints. As another example, the CoverDrop project (Chapter 6) tightly integrates the anonymity infrastructure and the applications with each other. In particular, involving many users, of which most will only ever send cover traffic, dictates that parameters must be carefully chosen to keep power consumption low and excess traffic small.

In this dissertation I argue that there are critical challenges (and opportunities) for the deployment of strong metadata privacy on mobile devices. I approach this claim through examination of the status-quo and building new solutions that I then evaluate in the context of smartphones—with a particular focus on technical limitations imposed by hardware, operating systems, and existing protocols. This work is constructive and yields building blocks that can be used for future research and real-world applications.

1.1 Publications

Some chapters in this thesis are based on work that I have written with others. While some papers are already published, others are currently under review.

- Daniel Hugenroth, Alberto Sonnino, Sam Cutler, and Alastair R. Beresford. **Sloth: Key Stretching and Deniable Encryption using Secure Elements on Smartphones**. Under review, 2023.

  The Sloth paper forms the basis of Chapter 5. I led the design of the protocol, implemented the prototype, conducted all experiments, performed the data analysis, and was the primary author of the text. My co-author Alberto was the main author of the formal proofs which are included in the appendices. Alberto, Sam, and Alastair contributed towards the development of the ideas and their presentation.

- Daniel Hugenroth and Alastair R. Beresford. **Powering Privacy: On the Energy Demand and Feasibility of Anonymity Networks on Smartphones**. In Proceedings of the 32nd USENIX Security Symposium (USENIX Security '23), pp. 5431–5448, 2023.

  The *Powering Privacy* paper forms the basis of Chapter 3. I conducted all experiments, performed the data analysis, and was the primary author of the text. Alastair contributed towards the development of the ideas and their presentation.

- Daniel Hugenroth, Ceren Kocaoğullar, and Alastair R. Beresford. **Choosing Your Friends: Shaping Ethical Use of Anonymity Networks**. In Proceedings of Security Protocols XXVIII: 28th International Workshop, 2023.

  The *Choosing Your Friends* paper forms the basis of Section 2.2. This paper has been shaped by all three co-authors as well as the discussion during the workshop.

- Mansoor Ahmed-Rengers, Diana A. Vasile, Daniel Hugenroth, Alastair R. Beresford, and Ross Anderson. **CoverDrop: Blowing the Whistle Through A News App**. In Proceedings on Privacy Enhancing Technologies (PoPETs), pp. 47–67, 2022.

  I joined the CoverDrop paper after my co-authors had conducted workshops and finished its distinguishing requirements analysis. My contributions span the protocol design, most of its implementation, and all mobile aspects of this work. After its publication, I have been working with a news organisation on implementing CoverDrop for real-world use. This work discovered novel challenges that were not previously considered. I discuss these alongside potential solutions in Chapter 6.

- Daniel Hugenroth, Martin Kleppmann, and Alastair R. Beresford. **Rollercoaster: An Efficient Group-Multicast Scheme for Mix Networks**. In Proceedings of the 30th USENIX Security Symposium (USENIX Security '21), pp. 3433–3450, 2021.

  The Rollercoaster paper forms the basis of Chapter 4. I led the design of the protocol, implemented the prototype, conducted all experiments, performed the data analysis, did the security analysis, and was the primary author of the text. Martin and Alastair contributed towards the development of the ideas and their presentation.

1.2 Contributions

In this dissertation I make the following contributions:

- In Section 2.2 I discuss ethical considerations of anonymity networks and their applications. A topic that still attracts little attention. I argue that specialised anonymity network designs can reduce the potential for harm and demonstrate this practically with the CoverDrop system.
- In Chapter 3 I discuss the energy consumption of anonymity networks on smartphones by measuring individual operations and whole protocols. For this I designed a hardware setup that also enables other researchers to conduct precise energy measurements easily and with minimal costs. Previous results for cryptographic operations on smartphones are outdated and I show that these no longer present a bottleneck. Also, I collect energy estimates for Tor, VPN, and Loopix in a comparable manner for the first time to show the importance of researching smartphone-friendly anonymity network designs.
- In Chapter 4 I present an efficient multicast scheme for mix networks, named Rollercoaster, that makes low-latency group communication possible—even in networks that rely on traffic shaping. For this, I built a simulator for the Loopix network to show that Rollercoaster reduces latencies for groups up to 100 members to make them practical for collaborative work. I also present a fault-tolerant variant that can handle offline users.
- In Chapter 5 I present schemes, called Sloth, that use the Secure Element for key stretching and plausibly-deniable storage on mobile phones. The Sloth schemes can be used by app developers on both Android and iOS without modifications to either hardware or software. I developed these based on my CoverDrop contributions. I also conducted a study that examines the prevalence of Secure Elements on modern smartphones as well as their API and performance.
- In Chapter 6 I discuss an anonymity network design, named CoverDrop, that allows whistleblowers to contact journalists securely. The CoverDrop project is joint work by many people over multiple years. I present challenges that arose from implementing CoverDrop within a news organisation and I extend the system design to support forward security, a more practical message scheduling strategy, and a plausibly-deniable private sending queue.

Chapter 2

Background

The following chapters in this thesis build upon the anonymity networks and technologies introduced in Section 2.1. In particular, the practical Loopix design (Section 2.1.4) will be central to the design of Rollercoaster in Chapter 4. In Section 2.2 we discuss some of the ethical implications that emerge from design choices of anonymity networks. Section 2.3 explores the global success of mobile devices and the status-quo of Android and iOS as the dominant smartphone platforms drawing attention to their unique challenges and opportunities.

2.1 Anonymity networks

Anonymity networks allow their users to exchange messages without revealing metadata such as who is talking to whom and when. Otherwise such information allow adversaries to learn about communication partners and groups as well as communication patterns. This metadata by itself is quite powerful as it could allow, e.g. targeting of activist groups by following communication traces originating at their leaders; or identifying the group of potential whistleblowers that had been in contact with a journalist just before a story was published.

In the threat model for anonymity networks we typically assume strong adversaries with resources similar to a nation state actor. Such an adversary has the ability to observe (encrypted) traffic on the network links between nodes. They are called a *local passive adversary* or *global passive adversary* (GPA) depending on whether they can only observe a subset of network links, e.g. only the direct connections to and from a user, or can see all network links. In addition, adversaries might compromise and control a fraction of the nodes that make up the network. Such compromised nodes can behave in a *Byzantine-faulty* manner, i.e. deviate from the protocol in any way that the adversary deems fit. This is different to *honest-but-curious* nodes where the adversary can only read their internal state, but the nodes continue to follow the protocol with no observable changes.

In order to communicate with each other securely over an anonymity network, both parties need to share their network addresses such that their messages can be routed to the intended destination and exchange their public keys to verify the authorship of messages from the other party. Most anonymity network designs assumes that this bootstrapping step happens out-of-band—either in-person or using an already trusted channel. Performing this exchange of information and discovering users based on

human-friendly identifiers inside an anonymity network remains an interesting problem without a satisfactory solution.

There are many different anonymity network designs and implementations that serve specific use-cases. As such they achieve different trade-offs between metrics such as latency which measures how quickly a message from A reaches B; bandwidth specifying how much data A can send per time unit; overhead in terms of computational costs and network usage; and anonymity guarantees describing how much information the adversary learns. The Anonymity Trilemma [46] shows that in fact no solution is able to tick all these boxes. In addition, individual anonymity network designs provide other benefits ranging from support for offline clients to censorship circumvention techniques.

2.1.1 Terminology and properties

The term *anonymity* is often used as a broad description for properties of communication system. However, using it without further specificity is not helpful as anonymity is an intrinsically relative property. It is only meaningful to describe the relationship between a party which tries to stay anonymous and a party that tries to identify them. Usually, we call the former the *user* and the latter the *adversary*.

In addition, we want to further qualify what exact information we try to hide from the adversary. For this we follow the terminology by Pfitzmann [108]. We use *anonymity set* to describe a group of subjects which all appear reasonably likely to the adversary as the target they try to identify. *Unlinkability* describes the inability of the adversary to generally associate subjects, actions, and objects with each other. For instance, what message was sent by whom. Finally, *unobservable communication* describes the strong property where the adversary is unable to determine whether a given subject is currently communicating any meaningfully information at all. This is typically achieved using *cover messages* which generate made-up traffic that is independent of user actions. The details of cover messages are discussed in Section 2.1.3. *Steganography* is a related property where the existence of any message, real and cover, is hidden. However, assuming that the use of anonymity technology is generally allowed, steganography provides little additional security benefit for the use-cases in this dissertation.

We use the term *strong metadata privacy* for anonymity systems where all metadata is hidden from the adversary. As such this term depends on both the system guarantees and the threat model. Typically, strong metadata privacy implies that anonymity networks use cover traffic to achieve unobservable communication against a global passive adversary.

2.1.2 Onion routing networks and Tor

Onion routing describes all protocols where messages are encrypted in multiple layers that are "peeled off" as the message travels along a number of intermediate nodes. Each hop decrypts the outer-most layer of the messages it receives and then learns about the address of the next hop and the encrypted messages to send there. As such, each intermediate node only learns its direct predecessor, from which they received the message, and successor, to which they send the encrypted inner message.

If the number of hops is just one, then this design assembles a VPN service where users tunnel their traffic through one of the provider's servers. In this case a local adversary that observes an

user's Internet connection can no longer read the intended recipient, e.g. a web server IP address, from the packet header. However, an honest-but-curious VPN server can easily record all metadata of its users, as they can see all routing information. Similarly, a GPA which observes the VPN server's traffic can correlate the incoming and outgoing packets and use that information to deduce which user communicates with which server. When increasing the route length to two, three, or more intermediate nodes, no single node can learn the entire path. In particular, such a setup hides the full route information as long as at least one intermediate node is not compromised by the adversary. We look at a popular implementation of such a design next.

Tor [50] is a circuit-based onion routing network and the most practical and widely deployed anonymity network today. Its intermediate nodes, called *onion routers*, are located in almost all countries around the world and Tor is being used by millions of users every day [131]. Routes in Tor consist of the guard node, a middle node, and an exit node and as such it remains safe even if two of the user-chosen intermediate nodes are compromised by the same adversary. In addition, to access to the Internet via the exit nodes, Tor also offers *onion services*. These onion services are only discoverable via their `.onion` addresses which allows other users to initiate a rendezvous process that establishes a connection between them and the onion service. Both parties remain anonymous towards each other and their traffic stays entirely within the Tor network. As these services are typically not accessible from outside Tor, they are sometimes referred to as the "Dark Web".

Tor is often used for web browsing and interactive communication. Therefore, its design treats performance, in particular low latency and high throughout, as important goals. For instance, messages are never deliberately delayed at onion routers to minimise latency, but this also makes it easier for adversaries to match incoming and outgoing packets. Moreover, once a circuit is established, all traffic to and from that user travels via the same route. This reduces the required routing information in the packet headers once the circuit is established, but at the same time it helps the adversary to identify messages that belong together. Maybe counter-intuitively, from an individual user's perspective, having a fixed circuit can be preferable over having many short-lived ones. This is because sending data over many independent ones increases the chance to create at some point a circuit that consists entirely of compromised nodes.

The fixed circuits, the low-latency message processing, and the lack of cover traffic make Tor vulnerable to traffic analysis attacks where an adversary tries to correlate users, messages, and streams. Murdoch et al. show that these attacks are even possible for adversaries that only have a partial view of the network [95]. Another form of attack from a (local) adversary are fingerprinting attacks based on the pattern of the encrypted traffic that a user receives. Wang et al. show the practicality of such attacks by identifying the websites that users are visiting with accuracy beyond 90% in realistic settings [145]. Similarly, peer-to-peer communication, e.g. between a user and an onion service, opens up correlation attacks as another traffic analysis attack vector. If an adversary suspects that A is sending messages to B, they can listen on the local network links of both users to see whether outgoing packets at A are followed by incoming packets at B shortly afterwards. By repeating these observations over multiple rounds, the adversary can build up confidence in their hypothesis.

2.1.3 Mix networks and Sphinx

Mix networks were one of the very first designs of anonymity networks proposed by Chaum [34]. On top of onion routing, they introduce two measures that help them resist attacks by a GPA. First, each message is routed independently which makes it more difficult to learn which messages belong to the same communication session between two users. Second, messages are delayed and mixed at the intermediate nodes so that it is hard to know which incoming message belongs to which outgoing message. For this, the mix nodes can employ different strategies. Threshold mixes wait until they have collected a set number of packets and then release them in a randomised order. Alternatively, timed mixes wait for a set duration before they shuffle and release all collected packets. While simple, both strategies can suffer from high-latency and varying anonymity guarantees based on the traffic load. Section 2.1.4 introduces a more favourable continuous-time mixing strategy where messages are delayed independently by sender-chosen durations.

As each message in a mix network is routed independently they require a packet format that efficiently store the entire routing information. In addition, both the header and payload of the packet need to be modified by each hop so that they appear completely unrelated. Otherwise, an adversary gains a benefit in linking incoming and outgoing packets at mix nodes which would defeat the randomised mixing with other packets. The Sphinx [44] packet format, which is also used by Loopix (see below), fulfils these requirements. In addition, Sphinx can carry auxiliary information, such as the delay durations for each hop, and it hides the number of remaining hops that the message still has to travel. The latter allows mixing of all packets, regardless of how far they still have to travel, to form a common anonymity set within mix nodes.

Over the years there have been multiple mix-based designs and implementations including Mixmaster [96] and Mixminion [43]. However, none of them achieved widespread and sustained adoption comparable to Tor. Loopix, which is introduced in the next section, stands out as a modern and practical design which is implemented and used in practise.

2.1.4 Loopix

Many parts of this thesis build upon Loopix [110] which we consider the most practical mix network design today. Most notably, it introduces provider nodes that mediate access to the network. In this role they can charge users for access and share the revenue with the operators of the mix nodes. In addition, providers also enable support for clients that are not continuously online by operating an inbox that caches incoming messages while the client is offline. Loopix uses a stratified topology where mix nodes are arranged in $l = 3$ layers such that messages from a mix node in layer i are always forwarded to a mix node in layer $i + 1$. The provider nodes are connected as inputs to the first layer and outputs of the last one. Figure 2.1 illustrates this topology. All messages are source routed, meaning that the sending client decides their full path information.

Each Loopix client sends three separate streams of outgoing messages which all use the same encrypted packet format ensuring that an observer cannot distinguish between them. Drop messages are regular cover traffic addressed to a randomly chosen provider node which will ignore them. Loop messages are cover traffic messages that are addressed to the sender itself. Hence, once such a loop message has been sent, the sender will expect to receive them back within a certain time frame.

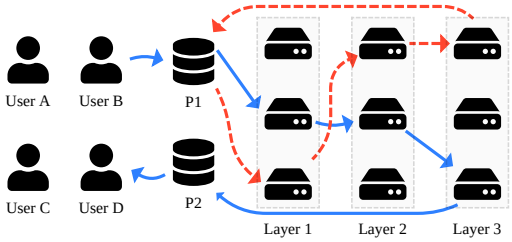

Figure 2.1: Overview of a Loopix network with four users (A–D), two provider nodes (P1, P2), and mix networks arranged in three layers. In this stratified topology each mix node in layer i is connected to all mix nodes in the subsequent layer $i+1$. The dashed red line indicates possible loop traffic generated by a mix node. The solid blue line indicates payload traffic from user B to user D. Own graphic from the ROLLERCOASTER paper [68].

Received loop messages—or the lack thereof—can be used to evaluate the health of the system and to identify misbehaving mix nodes. In Loopix, mix nodes also generate such loop traffic to continuously monitor the network. Finally, the client maintains a first-in first-out (FIFO) buffer for payload messages. If the payload buffer is empty when the next payload message should be sent, a new drop message is created as cover traffic and sent instead.

Parameters and stochastic modelling. We now discuss how the individual sending events and delays in the Loopix system are modelled. This is important background for both Chapter 3 and Chapter 4 and helps to build an understanding of the expected latency in the system.

The timings for the drop, loop, and payload streams are modelled as Poisson processes $Pois(\cdot)$ with rate parameters λ_d, λ_l, and λ_p respectively. This means that the delay between subsequent messages of a stream with rate $\lambda_{...}$ is sampled from the exponential distribution $Exp(\lambda_{...})$. For example, given the Poisson process $Pois(\lambda_x)$ with $\lambda_x = 2\,\mathrm{s}^{-1}$, one expects on average 2 messages per second and that the mean delay between two subsequent messages is $\frac{1}{\lambda_x} = 500\,\mathrm{ms}$. As the drop, loop, and payload streams are independent, the observable overall message stream of the client also forms a Poisson process $Pois(\lambda)$ with $\lambda = \lambda_d + \lambda_l + \lambda_p$.

For each outgoing message, clients sample the delays d_i from an exponential distribution $Exp(\mu)$ for the chosen mix node in each layer i and for the hop $(i = 0)$ from the sender's provider node to the first mix node. This Poisson mixing strategy allows for elegant stochastic modelling of the mix node behaviour and state. For instance, the total network latency of a message is the sum of all its inter-hop delays sampled from independent exponential distributions with parameter μ. This distribution is also known as the Gamma distribution $\Gamma(l+1, \frac{1}{\mu})$ which has the mean $\frac{l+1}{\mu}$ and where the shape parameter $l+1$ denotes the number of independent delays the message is experiencing.

When considering the total application-level end-to-end delay, one must also include how long an outgoing message msg has to wait in the payload FIFO buffer. Here each message has to wait for all previous n messages in front of it and their independent sending delays sampled from $Exp(\frac{1}{\lambda_p})$. Hence, this again yields a Gamma distribution $\Gamma(n+1, \frac{1}{\lambda_p})$ where the shape parameter $n+1$ accounts for all n previous messages plus the own delay of the message msg. Finally, messages are not directly

sent to clients, but to the inbox at their providers from which they are downloaded by the client. These downloads happen at regular intervals of δ_{pull} which implies that the mean waiting time for the next pull once a message arrives at a provider is $\frac{\delta_{pull}}{2}$. Combining the payload buffer waiting time, the delays during transit, and the time until the next pull yields the distribution of the total end-to-end message latency d_{msg}. With the aforementioned intermediate results, the expected mean message latency between two users therefore is

$$mean(d_{msg}) = \frac{n+1}{\lambda_p} + \frac{l+1}{\mu} + \frac{\delta_{pull}}{2}. \tag{2.1}$$

Loopix is compatible with a variety of parameters ranging from sending a few messages per minute to high rates where there are many messages per second. One limiting concern is the outgoing bandwidth which is $\lambda \cdot |msg| \frac{\text{bytes}}{\text{s}}$ where $|msg|$ is the fixed message size in bytes. Choosing very high rates requires significant traffic which can be costly. Another constraint is the relationship between λ and μ. If both overall the sending rate λ and the average delay $\frac{1}{\mu}$ are low, fewer messages are delayed in a mix node at the same time. Since there are fewer packets to choose from, this helps the adversary to guess which incoming packets relate to which outgoing packets.

With suitable parameter choices Loopix provides strong metadata privacy against adversaries that can observe all network traffic and compromise a fraction of mix nodes. In particular, it provides unobservable communication as the outgoing traffic does not allow to distinguish whether a client is sending real payload messages at any given time. As all traffic is generated by the client, this also holds when the sender's provider node is compromised. However, Loopix provides no receiver unobservability in case their provider is compromised. This is because providers can observe the total number of received messages for all their users. If one user receives more messages than the others, this indicates that they are likely receiving payload messages.

Nym. The Nym network [49] is an implementation based on Loopix that is deployed with hundreds of mix nodes across many countries [100]. Its default parameters are chosen for low-latency peer-to-peer communication with an average packet delay of $\frac{1}{\mu} = 50\,\text{ms}$ per hop, a payload rate of $\lambda_p = 20\,\text{ms}^{-1}$, and loop traffic rate of $\lambda_l = 200\,\text{ms}^{-1}$. Hence, the expected mean application-level latency is well below one second. However, Nym deviates from the vanilla Loopix protocol in two important aspects. First, clients do not send dedicated drop traffic (λ_d). Instead, they only send loop traffic (λ_l) and real or cover messages based on the payload stream rate (λ_p). Second, providers forward received messages without delay to the client as push-messages over a websocket connection. This improves the ability of an adversary to break receiver unobservability similar to the compromised receiver provider node mentioned above.

In Nym, users pay provider nodes with so called Nym tokens (NYM) for access to the network. These tokens are then pooled and can be earned by the operators of mix nodes. Users and operators can also delegate NYM to mix nodes as an endorsement of their reputation. Currently, the Nym client is only available as an open-source Rust implementation for desktop computers. However, there is no inherent technical limitation that prevents running such a client on a mobile device. I supervised a student project that ported the Nym client to Android and demonstrated its core functionalities to send and receive messages on the main network [84].

2.1.5 Other anonymity networks

In addition to the aforementioned anonymity networks, there are numerous other interesting designs and implementations. However, in general they have not been able to demonstrate adoption and practicality to the same extent as Tor and Loopix. In addition to the ones introduced here, the curious reader will find many more designs, historical development, and discussion in the systematisation of knowledge (SoK) paper by Sasy and Goldberg [117].

Dining Cryptographers (DC-Nets). A few years after introducing onion routing networks, Chaum published the problem of the Dining Cryptographers to motivate the construction of *DC-nets* as an alternative approach to anonymous communication [32]. In the original setting, a group of cryptographers sit around a round table and try to collectively determine a single bit answer, namely whether or not one of them has paid the dinner bill. However, being very privacy conscious individuals, they do not want to reveal which of them paid, i.e. ensuring anonymity of the sender. In Chaum's construction each of them first chooses their true answer, i.e. 0 if they have not paid and 1 if they have paid. If their true answer is 0, they sample a random coin flip and privately share this as the result with the persons directly left and right of them. If their true answer is 1, they do the same, but invert one of the bits that they share. Following this procedures everyone will receive the results of their left and right neighbours. Next, each cryptographer combines the received results in an exclusive-or (xor) operation and shares the resulting bit with the group. Finally, the group collectively xors all bits to determine the final answer. This works because the two inputs from the non-paying cryptographers will cancel out. If there was a cryptographer that has paid, they would have contributed just a single 1 which will then form the final answer.

While the original idea is intriguingly simple, it is not practical for real-world deployment. In the classical design all participants share their results with all other participants resulting in quadratic communication effort. Also, the protocol relies on synchronisation of all participants and the unavailability of a single person brings the entire system to a standstill. Similar to radio networks, participants must also agree on who is sending at what time so that communication does not interfere. Finally, a single participant can intentionally report incorrect results for their calculation which will alter the final outcome and cause system-wide disruption.

These challenges have been addressed in more involved, practical designs. Dissent [41] introduces a shuffle mechanism that assigns slots to individual users so that communication does not interfere. For improving scalability, D3 [149] and a later Dissent version [147] use servers that collect and distribute messages lowering the traffic to linear growth. They can also handle missing responses from clients. However, disruptions remain a problem and can only be dealt with through a blaming protocol that helps to find the disrupting participants retrospectively, but cannot prevent them from causing disruptions in the first place.

Private information retrieval (PIR). The research area of PIRs concerns techniques that allow clients to query data from a central database such that neither an observer nor the database learn which information they queried. A naïve implementation would have all clients download the entire database and then query it locally. However, this quickly results in excessive traffic. Instead, modern schemes [47, 61, 91] use homomorphic encryption techniques and optimised server code in order to

try to achieve low-latency responses with low communication overhead. These schemes typically find a meaningful trade-off between communication overhead, computation costs (client and server side), and security assumptions.

Pung [6] is a private communication network based on PIR where sender and receiver exchange messages by placing them under agreed-upon addresses on a central PIR server using a common shared secret. The SealPIR [5] approach improves the communication overhead and amortised computational costs allowing to improve on the network costs and throughput of Pung. Naturally, central addresses for message exchange lend themselves to sharing a message with many participants. The Talek [36] system uses this insight to implement multicast for groups.

Compared to mix networks, PIR-based anonymity networks are centralised and rely on the availability of a few powerful servers. However, they generally do not require any trust in the central nodes and allow for fully untrusted infrastructure. As all message writes and queries require significant processing costs from the central server, they have higher latency that increases with the total number of participants.

Mix networks. Besides Loopix, the mix network approach inspired many practical systems. However, they have not been deployed as systems that found real-world adoption similar to Nym.

Vuvuzela [139] first routes user messages through a mix network and then deposits them in a dead-drop that has been agreed upon by the communication partners during a dedicated dialling protocol. It is complemented by Alpenhorn [86] which is built upon the Vuvuzela infrastructure to allow looking up of participants using human-friendly identifiers. The Stadium [135] further develops the idea of Vuvuzela by introducing verifiable parallel shuffling that improves support for larger numbers of users while working with lower-spec servers. Groove [18] is a recent design that introduces "oblivious delegation". This allows them to tackle many practical challenges concerning modern usage patterns, such as supporting multiple devices per user and dealing with offline users. They also estimate the energy consumption of their protocol which we discuss in Section 3.1.4.

Notably, all mentioned systems are designed such that the noise, which is added to hide communication patterns, allows quantification of the achieved anonymity in terms of differential privacy guarantees. However, this comes at the cost of high message latency exceeding 10 seconds. Compared to Loopix or Nym, which support latencies of less than a second, this restricts their utility for interactive usage such as web browsing.

Other. The Invisible Internet Project (I2P) [151] is a peer-to-peer anonymity network where intermediate nodes are run by volunteers in a decentralised fashion with a central directory server. Instead users choose their peers from a locally maintained database that they update as they discover and measure potential peers. The I2P primarily considers only internal communication, similar to Tor's onion services, and does not provide access to the open Internet out of the box.

2.2 Ethical considerations for anonymous communication

This section is based on the paper "Choosing Your Friends: Shaping Ethical Use of Anonymity Networks" [69] that I have co-authored with Ceren Kocaoğullar and Alastair R. Beresford. For this

section I adapted the text from that paper to fit with the dissertation. While I proposed the initial direction and content, the final work is the result of equal contributions from all authors.

Anonymity networks, like end-to-end encryption, are a dual-use technology. As such they are, on the one hand, important enablers for positive use-cases ranging from protection of whistleblowers to counterbalancing mass surveillance that motivate research in this area. However, on the other hand, they can also be used by malicious actors for harmful activities. For instance, they might allow criminals to avoid detection or prosecution by law enforcement, make it easy to share misinformation and abusive messages, and provide infrastructure for running illegal online market places. The Silk Road was an infamous example of such an online market place that facilitated trade with illegal goods [37]. The Tor Project has an entry in their FAQ that addresses this conflict by arguing that criminals can already carry out their malicious acts by e.g. stealing smartphones or engaging in computer hacking [133]. In our opinion, this argument too generally assumes a motivated, professional, and well-resourced actor. Instead, we should also consider more opportunistic criminals which might actually experience an impact from not having an easy means of online anonymity which then could increase hesitation.

We believe that it is important to reflect on the potential use-cases when designing an anonymity network. In particular, the architectural and engineering design choices, of which we present some below, influence which use-cases an anonymity network is well suited for and which use-cases it does not support or is less feasible for. While none of these design choices can fully control how a network is eventually used, they can promote positive use-cases and disincentive unwanted behaviours.

2.2.1 Design decisions

One observation is that we can deliberately remove anonymity guarantees for certain relations between participants. For instance, in the CoverDrop system (Chapter 6) users can anonymously reach-out to journalists who will not learn which user sent a given message. However, the other way around, users always know from which journalist they received a reply. This enforced communication topology has another advantage. It prevents anonymous communication between users and hence effectively prevents activities such as running illicit online market places and sharing of media.

We can also consider technical limitations, such as latency and bandwidth, that influence the usage of an anonymity network. These limitations could be an otherwise undesirable performance characteristic of the system or chosen deliberately by the system designer. A high latency network will not be favourable for voice-over-IP communication and video meetings, while very low bandwidth makes the network less usable for sharing of media files. Similarly, the anonymity network can either enable access to other resources, such as Tor allows accessing regular websites, or provide internal endpoints such as the Tor onion services. The former provides more immediate value for users and enables, e.g. law enforcement, to take action against servers that host illicit content; where as the latter, i.e. Tor onion services, generally come with stronger anonymity guarantees and cannot be easily located.

Finally, content and contact discovery have a great impact on the supported use-cases and potential for abuse. This stems from the observation that feed-based communication networks such as the social media platforms TikTok and Facebook have been suspected to help amplify misinformation and controversial messages [62, 104, 127]. On the other hand, system that require one-to-one contact

establishment through out-of-band contact exchange, such as Signal, are less prone to these use cases. The need for interactions to setup these channels limits how easily an actor can increase their reach in these systems. A third category are networks that have a shared index that allows discovery of content based on keywords or similar filters. We think that these lend themselves to be used for sharing illicit media content or facilitating online market places.

2.2.2 Related work on the ethics of anonymous communication

The privacy guarantees of anonymity networks make it, as it is intended, hard to study how they are being used by real users. However, operators of Tor entry nodes can distinguish whether a user is accessing Tor onion services or regular Internet pages. The relative prevalence of these connections types is used by Jardin et al. as a proxy metric in their study [73]. They further assume that regular websites are less likely to host illicit material since hosting it on a regular server allows law enforcement to discover its physical location and the responsible administrators. Analogously, the less discoverable onion services are assumed to more likely host bad content. Based on their data they make the observation that users in "non-free countries" are more likely to access regular websites; hence, using Tor to circumvent potential censorship and hiding their legitimate usage from adversaries. Whereas users in "free countries" are more likely to access onion services indicating that Tor is more often used for accessing (illicit) services that would have been banned if hosted on regular servers.

Discussions of pseudonym versus real-name policies on social media platform are an insightful resource since in these discussions the use of pseudonyms is often effectively equated with anonymity. In his essay Bodle summarises ethical considerations related to pseudonyms on social media platforms and argues that they are "indispensable as an enabler of other inalienable rights" [25, p.22] and can encourage "honest self-disclosure" [25, p.26]. He concludes that a simple utilitarian perspective is insufficient as not all consequences are foreseeable; and instead the issue should be addressed with an ethical pluralist approach.

Finally, there are considerations regarding trust alignment for peer-to-peer (P2P) networks. Danezis and Anderson [42] look at file-sharing networks that operate in a P2P architecture and distinguish between a "random model" where content is indiscriminately stored on random nodes and a "discretionary model" where content is only stored and shared by those who also consume it. Similar analysis would be interesting for anonymity networks as well where we could substitute the goal metric of availability of content with effective anonymity.

2.3 Mobile devices

Mobile devices have become the primary computing devices for many around the globe. For this we first explore how people access the Internet and discover the important role mobile devices play for connecting the world. We then briefly introduce the two dominant smartphone platforms, Android and iOS, and their respective ecosystems. This provides the opportunity to contrast smartphones with desktop computers and see what impact these differences can have on deploying secure applications and protocols. Finally, we highlight the limited energy supply as one of the defining characteristic of mobile devices which we then discuss in the following chapter.

2.3.1 Access to the Internet

Mobile devices have become the dominant computing devices through which people around the world access the Internet. This becomes apparent when exploring the statistics [130] published by the International Telecommunication Union (ITU), which is the United Nations (UN) specialised agency for information and communication technologies (ICTs). We present an analysis of this data for Internet access reach and mobile connectivity in the following paragraphs. As more than 2 billion people remain without access to the Internet [129], it is particular interesting to look at countries who are currently connecting many of their inhabitants for the first time. For this we discuss numbers for both global averages (World) and for the Least Developed Countries (LDCs). The list of LDCs[1] is published and reviewed by the Committee for Development Policy (CDP) which is part of the United Nations Economic and Social Council (DESA) [137]. LDCs are defined as "low-income countries suffering from structural impediments to sustainable development." [136, p.3]

First, we are interested in estimating the population reached by both fixed-broadband and mobile Internet access. The ITU statistics [130] provide subscription numbers per 100 inhabitants aggregated (amongst other categories) for LDCs and all countries. We assume that mobile subscriptions are typically accessed by one individual while fixed-broadband access is shared with members of the same household. The UN's Population Division publishes a dataset of household compositions [138] that we use to calculate the average household size world-wide (4.18 members) and in LDCs (5.51 members). We multiply the fixed-broadband subscription numbers by the respective household size to estimate the reach. Figure 2.2 plots the estimated reach for data available from 2005. On a global scale, fixed-broadband subscriptions were the primary way to access the Internet until around 2016 when the reach of mobile access overtook it. For LDCs, fixed-broadband subscriptions were never widely adopted, but mobile subscriptions have grown to over 40% reach within the last 10 years. This strongly suggests that for many people in these countries, mobile devices provide the first and only opportunity to access the Internet.

Second, we are interested in estimating the quality of mobile Internet access. The ITU statistics [130] provide numbers that describe the relative percentage of the population reached by different mobile phone network standards. The dataset distinguishes between basic "mobile-cellular" connectivity, "at least 3G", and "at least 4G". We use this as a proxy metric for available bandwidth and latency and assume that 4G generally allows using most applications and services available on the Internet. Figure 2.3 plots mobile connectivity data for the last 7 years. In the global aggregate almost everyone appears to be covered by some mobile network and more then 80% have access to modern 4G networks. For LDCs, the coverage by some mobile network is still high, but half of the people in these countries do not have access to 4G networks. This means that modern Internet applications such as online video conferences are not available or only available with degraded user experience. These numbers are biased towards urban areas where it is easy to connect many people with relatively little infrastructure.

[1] At the time of writing, the list of LDCs includes 46 countries: Afghanistan, Angola, Bangladesh, Benin, Bhutan, Burkina Faso, Burundi, Cambodia, Central African Republic, Chad, Comoros, Congo, Djibouti, Eritrea, Ethiopia, Gambia, Guinea, Guinea-Bissau, Haiti, Kiribati, Lao People's Democratic Republic, Lesotho, Liberia, Madagascar, Malawi, Mali, Mauritania, Mozambique, Myanmar, Nepal, Niger, Rwanda, Sao Tome and Principe, Senegal, Sierra Leone, Solomon Islands, Somalia, South Sudan, Sudan, Timor-Leste, Togo, Tuvalu, Uganda, United Republic of Tanzania, Yemen, and Zambia

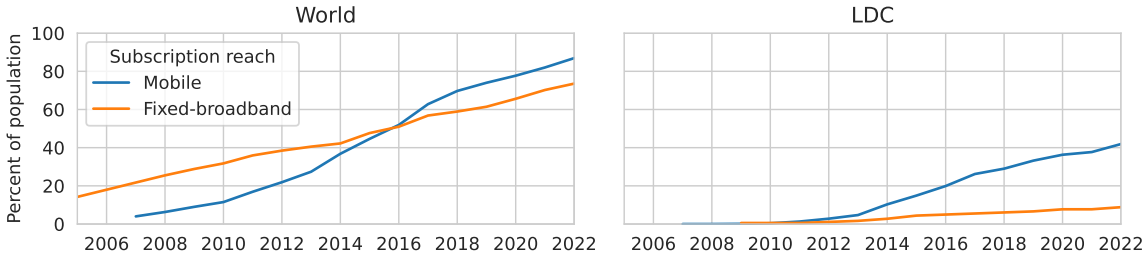

Figure 2.2: Estimated relative population reached by mobile and fixed-broadband Internet subscriptions. The left plot shows world-wide averages while the right plot only includes countries from the UN's Least Developed Countries (LDC) list. The numbers for fixed-broadband reach assume that one connection is shared among all members of the household.

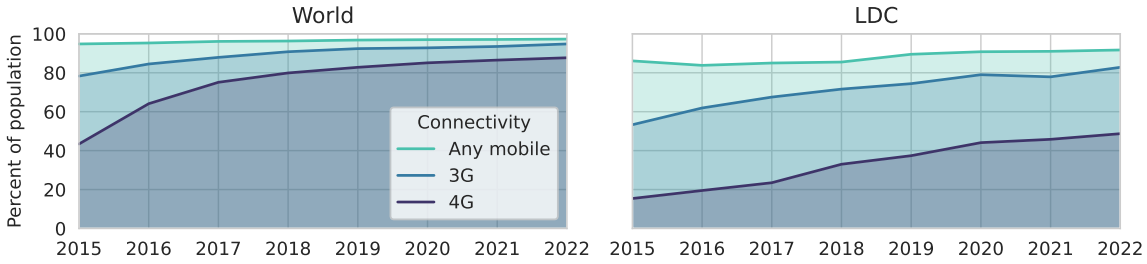

Figure 2.3: Estimated relative population that lives in an area covered by mobile networks of different generations. The left plot shows world-wide averages while the right plot only includes countries from the UN's Least Developed Countries (LDC) list.

As Doreen Bogdan-Martin, Director of the ITU Telecommunication Development Bureau, points out: "Those who are still not using the Internet will be the most difficult to bring online." [129]

2.3.2 Smartphone platforms

The smartphone market is dominated by two mobile operating systems. Apple's iOS platform accounts for 27.9% of all mobile web traffic in the first quarter (Q1) of 2023, while Google's Android accounts for 71.4% [125]. This leaves less than 1% for all other platforms rendering them negligible. However, these numbers vary between different regions. In North America the market share of iOS is 54.4% (Android 45.2%), whereas in Africa iOS only accounts for 13.6% (Android 83.9%) [125]. As such we can assume that populations who connect to Internet for the first time most likely do so on an Android device.

Apple keeps their proprietary iOS platform exclusive to its own end-user devices that include iPhones and iPads. As such Apple maintains complete control over both hardware and software allowing them to be tightly integrated. Google pursues a different strategy with Android and operates it as an open platform with its source code freely available. This allows other companies, such as LG and Samsung, to use Android as the operating system for their smartphones and to customise the entire system further based on their needs. As such the Android ecosystem is characterised by a

high-level of fragmentation of both device hardware, ranging from entry-level to high-level smartphones and platform software. For the lowest entry-level devices, Google provides a dedicated GO version that runs with less memory and storage [59].

On both Android and iOS users can install additional apps through software marketplaces. However, before users can download a developer's new app through the App Store (iOS, Apple) or Play Store (Android, Google), these new apps have to be approved by the marketplace operators. This centralised software repository promises app developers an easy distribution channel and users protection from malicious software. In exchange, the marketplace operators take up to 30% of the resulting revenue as their fees. The powerful position of these companies that control access to a large digital market through these marketplaces has raised concerns and led to the opening of investigations by both the UK's Competition and Markets Authority and the European Commission [39, 51].

Compared to other common end-user operating systems on desktop computers and notebooks, such as Windows, macOS, and GNU/Linux, mobile operating systems offer practical security advantages. Both Android and iOS implement strong isolation of individual apps where access to stored data, camera, Internet, and other resources is mediated through the system's permission manager. These permissions are either given statically during installation or must be requested from the user through a system-controlled dialog. This can protect users from malicious software and limit tracking and privacy invasion from otherwise trusted applications.

Many modern smartphones also come with integrated biometric authentication, such as fingerprint readers or face recognition, and allow full encryption of the device to preserve confidentiality of information if the device is lost or stolen. Often smartphones also come with dedicated hardware chips, so-called *Secure Elements* (SE), that can store non-extractable key material, perform cryptographic operations within its dedicated hardware, and maintain strong security guarantees even from physical attacks. The operating systems can use SEs to effectively limit attempts to unlock the device, securely store biometrical data, and attest that the device is running supported software. For app developers access to these SEs is only possible through a limited API. With Sloth (Chapter 5) we show how developers can still use it for building a strong key stretching scheme in user-level apps.

2.3.3 Energy consumption

Most user will expect their smartphone to operate for a full day. As such, the smartphone must carefully manage the limited energy stored in its battery. We assume that most users will charge their smartphone overnight and then use it for 12 or more hours before being able to conveniently charge it again. As battery capacity differs between different models, we assume a standard battery capacity of 8 000 mWh for our calculations. This is a typical battery size found in a range of mid-to-high-end smartphones [134].

Energy is typically given in millijoule ($1\,\mathrm{mJ} = 1\,\mathrm{mW} \cdot \mathrm{s}$) to describe the total cost of an operation from start to finish. When talking about continuous usage (e.g. when running a security communication protocol throughout the day) we are instead interested in power consumption, which is typically given in milliwatt ($1\,\mathrm{mW} = 1\,\mathrm{V} \cdot \mathrm{mA}$). However, these units can be hard to interpret and it is difficult to see what they effectively mean for the user experience. Therefore, we translate these numbers into percentage points (pp) of the total standard battery capacity. For example: if the battery is

drained by 4800 mWh then this corresponds to 60 pp. If this happens over a 12 hour period, then the respective power consumption would be 5 pp/h (400 mW).

The hardware of modern smartphones supports low-power idle states where power consumption is minimal, allowing the smartphone to achieve long standby times. As such the platforms are interested in reaching these idle states quickly (racing-to-idle) and maximising time before the next wake-up event. These wake-up events include incoming network traffic, user interactions, or background work by apps. Both platforms restrict when and how apps can perform work in the background while the app is not being actively used. By imposing a central API they can schedule background tasks of many apps in parallel so that the time in wake state is minimal and completely block apps that the user has not used recently. We discuss details of such mechanisms on Android in Section 3.1.3. To nevertheless allow timely notifications, e.g. for incoming email, both platforms offer a push-notification service that multiplexes data for all installed app. The app's backend will send a notification object to the respective server for Android or iOS and this will perform the delivery to the end-user device.

2.4 Summary

In this chapter we gave an overview on anonymity network designs, focusing on those which have been deployed in practice. We saw that Tor, the most widely used anonymity network today, works well against local adversaries, but is vulnerable to traffic analysis. Mix-network based designs using cover traffic, such as Loopix and Nym, are more resistant against such attacks and provide strong metadata privacy even against global adversaries. While there are alternative designs, these have not found widespread adoption.

While anonymity networks can be a force for good and a powerful tool to resist mass surveillance, they are inherently a dual-use technology that can also provide benefits to criminals. We briefly reflected on ethical considerations in this context and saw that design decisions can make anonymity networks more suitable for positive use-cases and less suitable for malicious usage.

We then looked at how people access the Internet and see that mobile connections dominate. This is particularly true for those who connect to the Internet for the first time in some of the least developed parts of the world. The smartphone market is dominated by the two mobile platforms, iOS and Android, which bring new security features such as strong app isolation, biometric authentication, and dedicated secure chips.

Finally we set the scene to talk about the energy consumption of mobile applications and introduce our metric of battery percentage points (pp). This leads us to the next chapter where we explore the energy consumption on Android smartphones in more detail. In particular, we perform measurements of both cryptographic operations and radio transmissions. We also examine how popular anonymous communication technologies fare in this context.

Chapter 3

Understanding the energy efficiency of anonymity networks on smartphones

While there are many interesting designs for anonymity networks (Section 2.1.5), there has been little consideration for the special circumstances when running them on mobile devices. Therefore, they overlook the critical challenges of intermittent connectivity and limited energy supply on mobile devices. While the former has been addressed in part by existing protocols, e.g. Loopix [110] and Groove [18], the latter has received little attention. We believe an anonymity network is not practical if it drains the battery too quickly. Therefore, an evaluation of energy consumption is crucial if we are to bring the benefits of anonymity networks to everyone.

Among the existing energy measurement studies, we did not find any that discuss and compare anonymity networks in detail. Those who cover individual operations of interest, such as cryptography and radio communication, were done on smartphones that are many generations old. We found that they do not translate to modern smartphones with better hardware and specialised instruction sets.

The current lack of evaluations can be explained by the complexity of mobile energy consumption and the measurement thereof. First, individual components such as the radio modules contain many internal states. Each data transfer (no matter how small) will incur costly state promotion from *idle* to *connected* where it will stay for a while in anticipation of subsequent transfers. Therefore, the bottom-line energy costs of a data transfer is also influenced by operations before and after. This makes attribution to individual apps and protocol steps difficult. Second, because these effects are global to the device, the operating system coordinates background work to maximise idle time and tries executing background tasks in batches to make the best use of the radio states. This means that we cannot examine just an app, but need to include the interplay with the surrounding infrastructure. Third, hardware-based power measurements often require extra equipment and expertise which make evaluations appear expensive. We believe that the system proposed in this chapter can make such measurements more accessible.

This chapter is based on the paper "Powering Privacy: On the Energy Demand and Feasibility of Anonymity Networks on Smartphones" [67]. For this chapter I adapted the text, figures, and plots from this paper to fit with the dissertation. I conducted all experiments, performed the data analysis, and was the primary author of the text. Alastair contributed towards the development of the ideas and their presentation.

3.1 Measuring mobile energy consumption

Smartphones are ubiquitous and we expect them to do more and more. Therefore, good battery life is key to their utility. We first introduce the general approaches to measuring the energy consumption of smartphones and then provide the background relevant for the energy measurement of radio transmissions, cryptographic operations, background scheduling, and anonymity networks.

3.1.1 Hardware-based and model-based approaches

One can divide the research field of mobile devices and energy consumption based on measurement approach and scope. The former comprises of hardware-based measurements and model-based approaches that are discussed below. Each of these approaches can be used to examine either individual operations (micro study) or more complex scenarios (macro study).

For *hardware-based measurements* researchers place a power monitor between the power source and the mobile device. They then execute an operation under test while recording the power consumption. Integrating over time (i.e. calculating the area under the curve) yields the total energy. Hardware-based measurements provide ground-truth results and reflect actual real-world power consumption. Also, hardware-based measurements do not influence the measurements as no additional debugger or tracing software runs on the device. On the downside, they do not attribute the energy consumption to individual components—say measuring only the CPU while ignoring radio communications.

Hardware-based setups are often used to capture individual operations. The work by Carrol et al. [30] and Adrito et al. [15] cover many basic operations from display illumination to phone calls. Other work examines specific areas such as WiFi [114], 4G radio communication [65], and machine learning [90]. The GreenMiner [63] project executes pre-recorded app interaction sequences for regression tracking. However, the mentioned existing work concerning the energy consumption of algorithms and radio communications lags multiple generations behind smartphones that are in use today. The recent BatteryLab project [140] focuses on providing remote hardware-based measurements at multiple locations. However, through the more general setup they are less suitable for our research as they provide limited control over the device under test and do not provide the high-resolution synchronisation required for analysing short operations. In terms of equipment, previous work uses either general-purpose power measurement appliances (e.g. Monsoon[1]) or build custom tools using power sensor chips like we do.

On the other hand, there are *model-based measurements* where researchers first create models based on execution traces and then use these to predict the energy costs when running other apps in a second step. These models are simple and effective when the power consumption is dominated by

[1] https://www.msoon.com/high-voltage-power-monitor

computation as CPU energy consumption is well-documented and their state (e.g. adaptive frequency) can be recorded cheaply. However, radio communications are harder to model as the radio module operates in different states (e.g. connected, idle) with transition latencies depending also on the activity of other apps. In general, models are bound to a specific training device and therefore the model prediction becomes out-dated together with that device. Their utility is further limited as they often need modifications to the apps or operating system (rooting), or use APIs that are no longer present in newer devices. The removal of APIs is often performed by the operating system vendor to reduce side-channel attacks and improve user privacy. For example, access to `/proc/stat` was removed [53] in 2017 citing an attack that exploited interrupt information to recover user input [123].

We found that model-based approaches are typically used to capture complex scenarios that last multiple seconds. PowerTutor [152] models the overall energy consumption of components such as CPU, radio, and display individually. This allows it to achieve high accuracy even for complex scenarios that include GPS and radio usage. However, the calibration devices are now more than 10 years old and many of the required APIs are no longer available. As radio communication is the main driver for many applications, the EnergyBox [143] project from 2014 focuses solely on predicting WiFi and radio energy consumption based on network packet captures. This was successfully used to examine mobile messaging applications [142] and specific mobile Tor usage [81]. Similar to other model-based approaches, EnergyBox suffers from calibration to older devices and protocol versions.

3.1.2 Cryptographic operations

Our literature review found no recent micro studies examining cryptographic operations on smartphones. However, such data is important for the design of new protocols as anonymity network implementations make heavy use of encryption, signatures, and key exchanges. The work by Potlapally et al. [112] and Rifà-Pous et al. [115] are some of the first to investigate both individual cryptographic operations and protocol executions on mobile devices (PDAs between 2006 and 2011). Montenegro et al. [92] compare the relative energy consumption of different cryptographic libraries on Android. However, since they use the PowerTutor model, their absolute energy predictions refer to Android devices that are more than 10 years old. The most-recent measurement of cryptographic operations that we found is a study of Elliptic Curve Cryptography (ECC) on an ARM-based Internet-of-Things (IoT) platform [94].

Radio operations

Radio communication is an intrinsically hard area for energy studies due to large numbers of internal states, delayed state transitions, and inter-application effects. When a mobile device starts to communicate via a mobile internet connection (e.g. 4G), the radio module will first promote the system from an idle state (with low standby power consumption) to a connected state (that requires more power to maintain). The transition itself costs energy and time, which is why the device will remain in the connected state for a while in anticipation of a response or more data to send. This is referred to as tail latency and its duration depends on the protocol and the mobile provider's network configuration. A delay of multiple seconds is typical. As a result, sending two small packets 30 seconds apart requires more energy than sending one very large packet without interruption.

Name	Ref.	Year	Anonymity	Bandwidth	Latency	CPU	Mobile Devices
Dissent	[148]	2012	◐	●	●	●	○[1]
Vuvuzela	[139]	2015	●	●	●	●	○
Hornet	[35]	2015	●	◐	●	◐	○
Riposte	[40]	2015	◐	●	◐	●	○
cMix	[33]	2017	●	○	●	●	○
Loopix	[110]	2017	●	●	●	●	◐[2]
Groove	[18]	2022	●	●	●	●	◐

Table 3.1: Summary of evaluation metrics used in recent anonymity network papers (● = thoroughly covered, ◐ = covered, ○ = not covered); [1]mentioned as important future work; [2]offline support.

The work by Huang et al. [65] examines 4G and its power consumption in great detail. Pathak et al. [103] are able to attribute energy costs to individual components, e.g. assigning the resulting radio energy to the background service that caused the transition to the connected state in the first place.

3.1.3 Android background scheduling

The operating system plays a key role in managing battery life, as the expectations of end-users and the number of applications grow. One of the most crucial aspects is the coordination of background services and tasks. Android 6 introduced *Doze* and *App-Standby* for this purpose [54]. Doze pauses background execution when the device is idle except for scheduled maintenance windows. By limiting the execution of background tasks to these windows, Android minimises the number of the state transitions of the radio module. App-Standby further reduces access to background execution for apps by learning which apps are used regularly by the user and placing restrictions on all inactive applications. When applications are found to be rarely used, the system might defer their background tasks by up to 24 hours or restrict background network access [55].

3.1.4 Anonymity networks

We reviewed the evaluation methods used in widely-cited and claimed to be practical anonymity network designs. The results are summarised in Table 3.1. We limited the list to recent publications which we would expect to cover mobile devices. Most work focuses on the bandwidth and achievable performance in terms of throughput and latency. The required computational cost (i.e. CPU) is often only considered to the extend necessary to rule it out as a potential bottleneck. PIR-based anonymity networks (Section 2.1.5) are the exception in this regard as they typically require costly computation from both the clients and servers. Most publications also quantify the achieved anonymity in terms of either anonymity set size or entropy.

Overall, there is little consideration for the practicalities of running these networks on mobile devices. Notable exceptions are Groove [18] which measures its power consumption, Loopix [110] which provides support for offline clients, Dissent [148] which flags this as important future work,

and Hydra [118] which highlights the difficulties of precisely scheduling background execution on Android (Section 3.1). Groove is the only one to provide measurements of its energy consumption on a smartphone. Unfortunately, their methodology limits the comparability and reproducibility of their results. Their setup measures the charging rate of the device instead of the power drawn by its components. In addition, apps and OS might undertake more work when they detect the device is charging. Their measured idle power consumption of 310 mW makes the discrepancy clear as it is much higher than a typical smartphone idle consumption of <100 mW. Also, their sampling frequency (once per second) is too low to capture peaks from cryptographic operations and radio transmissions.

3.2 Measurement methodology

We use a hardware-based approach for our measurements for three main reasons. First, we did not find any model for recent smartphones that covers both CPU and radio communication. Second, in our case studies we are interested in real-world battery life which must include the interference with the operating system scheduler, all utilised components, and side-effects. Third, we are interested in overall execution profiles that include the lowest standby states. Running software-based debuggers or tracers in the background would prevent the smartphone from entering these.

In our setup we install a Texas Instruments INA219 [126] power sensor between the smartphone and its battery. This sensor records 2 000 power measurements per second with 1% accuracy using a shunt resistor. An Arduino polls these samples via an I^2C bus and forwards them via USB to a computer that records timestamps and power values into a power measurement CSV file.

For the macro studies that run longer than a few seconds this setup is already practical as the operator can start and stop the measurement manually. However, for micro studies where individual operations only run for a few milliseconds we automate this process by creating the custom app *EnergyRunner* that executes the individual operations and records timestamps in an *execution log* on the device. We synchronise the time on the smartphone and the power measurement logs using a USB-to-Serial dongle (FTDI FT231X) that is connected to the smartphone's USB port. Its ground is connected to the common ground of the Arduino and one serial control line (e.g. RTS) is connected to a digital input port of the Arduino. The Arduino reports changes to the digital input via the same USB connection that is used for the power measurement samples. The synchronisation sequence is executed before the experiment so that we can later correct for differences in *clock offset*. I found that differences in *clock speed* are negligible (less than 0.001 ms per hour). The USB dongle is disconnected before the measurement of the actual operations start to ensure that its own power consumption does not influence our measurements.

Our choice of smartphones and hardware allows the tests to be done without permanent hardware modification and requires minimal technical skills. This is especially true since the battery can be removed without tools and placed into a custom 3D-printed battery holder. The full hardware setup is illustrated in Figure 3.1.

We now describe the protocol for running an experiment with our setup. First, the instrumentation app EnergyRunner loads a scenario file which describes the order and parameters of the operations to execute. It automatically adds the synchronisation sequence at the beginning of the execution schedule. We first start the recording software on the PC and then start the execution on the smartphone. After

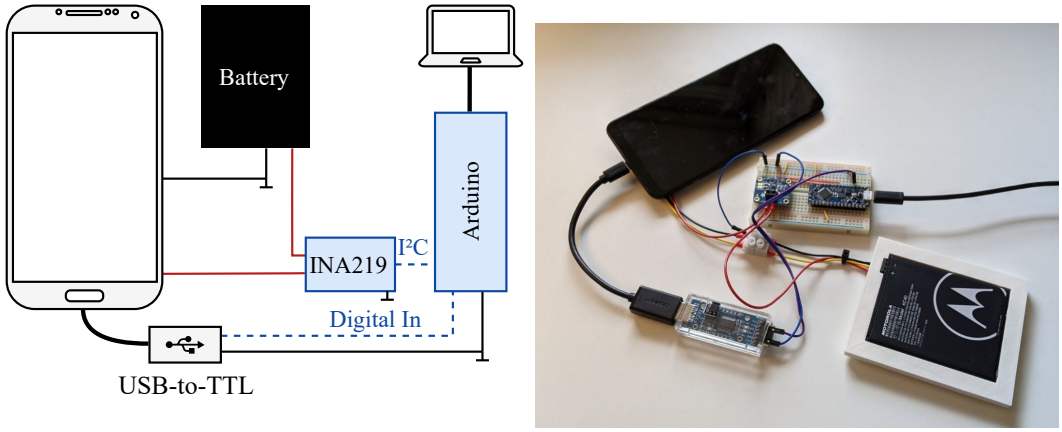

Figure 3.1: Schematic and photograph of our power measurement hardware setup.

the synchronisation phases has finished, we remove the USB dongle. During execution, EnergyRunner records the timestamps of operation start and end. Pauses between individual operations are described in the scenario file as well. After the entire execution schedule finished, the execution log with the timestamps is saved and uploaded. Finally, processing scripts will read both the execution log from the Android device and the power measurement CSV file. The processing code identifies the synchronisation patterns to adjust for clock differences and then extracts the power measurements for each individual operation based on the timestamps in the execution log.

We prepare the device under test by uninstalling and deactivating all apps that can cause background activity, such as Google Play Services. With the start of the execution of the scenario files the display and all radio connections are turned off unless needed by the operations under test.

Using this setup we can test individual operations semi-automatically, efficiently, and accurately. As we hope that such energy measurements become more common in papers that introduce (mobile) protocols, we put a lot of effort into making sure that this setup can be easily replicated by other researchers. All of our software for executing operations, logging the data, and analysing the results is available as open-source at `https://doi.org/10.5281/zenodo.10679431` under an MIT license. This also includes an interactive logging tool that shows incoming data in a live plot. Likewise, all our hardware specifications, 3D-printing files, and assembly instructions are part of the repository. The used hardware components are widely available, cheap, and easy to assemble. This work was submitted to the USENIX Security artefact evaluation process and was awarded the “Artifact Available” and “Artifact Functional” badges.

All studies are performed using the setup described in this section and use a Motorola Moto E6 Plus (released September 2019) smartphone running Android 9. It comes with an MT6762 Helio P22 chipset, a 2.0 GHz Cortex-A53 CPU, and 2 GiB RAM. We updated the smartphone to the most recent OS update and uninstalled and disabled all other applications.

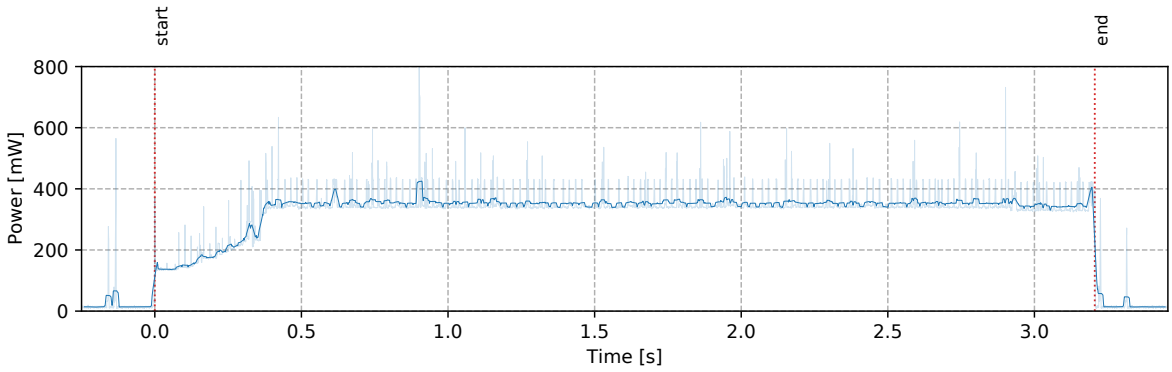

Figure 3.2: Power trace of generating a 2048-bit RSA key pair. The power rises between 0.0 s and 0.4 s as the system increases the core frequencies. The light blue line shows the raw measurements while the dark line is a rolling average.

3.3 Micro studies of individual operations

We start our evaluation by studying individual cryptographic operations (Section 3.3.1), background scheduling (Section 3.3.2), and radio transmission (Section 3.3.3). These are the important building blocks for anonymity network protocols and help interpreting the results of the macro studies.

3.3.1 Cryptography algorithms

This micro study evaluates to total energy (mJ) impact of individual cryptographic operations. We consider an operation negligibly cheap if their energy impact is below 4 mJ which is equivalent to more than ten million executions with a single battery charge. The results are summarised in Table 3.2.

Asymmetric cryptography is frequently used in anonymity network protocols for signatures and encryption of the payload. At the same time it has a reputation for being computationally intensive. Many anonymity networks use a message-based architecture (as opposed to a channel based one) where typically no key agreement between communication partners occurs. This means that every message that is sent or received involves asymmetric operations. Also, in multi-hop based architectures encrypting messages for each hop along the path further increases the number of asymmetric operations.

RSA is an established asymmetric cryptographic scheme that is still in common use. Its key generation algorithm involves finding prime numbers which can lead to long runtimes (Figure 3.2) and high variance due to non-determinism. Key generation for 4096-bit RSA can require up to 2 800 mJ which translates to just 14 500 executions with one battery charge. However, once a key is available, sign and verify operations are cheap. As RSA implementations use a high exponent for the private key and a small one for the public key, verify operations are much faster. If the exponent sizes are swapped (e.g. DSA), signing becomes cheaper.

In the last 20 years RSA has been slowly phased out in favour of Elliptic Curve (EC) cryptography. Its smaller key sizes generally allow fast execution, compact representation, and it does not require expensive prime-number search for key generation. We find that all operations are (negligibly) cheap and more efficient than those of equivalent RSA key sizes (256-bit EC is considered equivalent to 3072-bit RSA). We tested the four different curves commonly available on all Android versions through

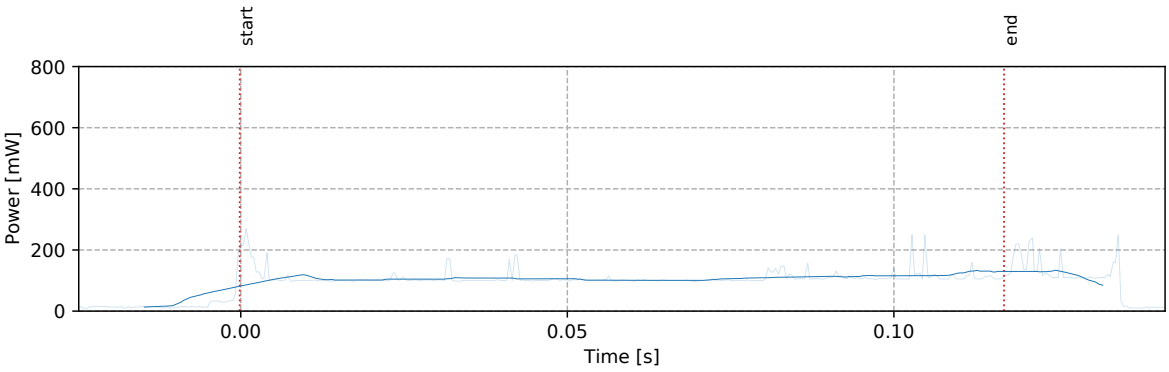

Figure 3.3: Power trace of a single Sphinx packet creation execution including JNI overhead. The light blue line shows the raw measurements while the dark line is a rolling average.

the built-in Android OpenSSL provider [66]. The `prime256v1`/`secp256r1` cipher is notably faster than the others.

It is possible to build a packet format for mix networks using these EC operations. However, actual implementations use specialised cryptographic constructions such as Sphinx [44] which allow for more compact packets and additional security properties. We evaluate a modern Rust implementation of Sphinx [101] (git commit: `c494250`) that is used in the commercial mix network Nym [49]. For our experiments we add a JNI binding to the library to allow calling it in our Android app. Since the JNI boundary adds a non-negligible overhead for such small operations, we add an iteration parameter so we can execute multiple rounds without leaving the native code. Each round calls `SphinxPacket::new()` which internally uses `Curve25519` primitives. The results suggest that this Sphinx implementation is efficient (Figure 3.3) and comparable to a few EC operations. The JNI overhead is up to 15% based on the difference between the configurations with 1× and 100× iterations.

The results show a significant reduction in energy costs compared to two other studies that use a hardware-based approach. Rifà-Pous et al. measured RSA and EC operations on PDA devices in 2010 [115], and Mössinger et al. measured EC operations a ARM-based development board in 2016 [115]. Compared to our results, the energy costs of the RSA operations on the PDAs are higher by factor ×3 (Verify RSA-1024) to ×20 (Sign RSA-2048). Operations like the generation of 2048-bit RSA keys were too resource consuming for the PDA and sometimes did not finish. The costs for EC-224 operations on the PDAs are higher by factor ×30. The ARM-based board using the optimised MicroECC library requires around ×20 times more energy than our test device.

We found that all standard hash operations are negligibly cheap. This is due to the low computational complexity and the wide availability of specialised CPU instructions such as `SHA256H2` and `AESE` on Arm64 [16, p.1556]. For completeness we also measured the energy required to hash a 16 KiB byte array with SHA-256/512 resulting in energy costs of 0.26 mJ and 0.27 mJ respectively.

3.3.2 Background scheduling

This micro study quantifies mechanisms for executing code while the phone is not actively used. These allow applications to perform message synchronisation and background computations. Anonymity networks use them for sending cover traffic during idle mode. This is critical for hiding whether a user is

Operation	Energy [pp]	Energy [mJ]	StdDev
Gen RSA-1024	0.000 404	116.47	76.36
Gen RSA-2048	0.002 764	796.05	597.84
Gen RSA-4096	0.010 064	2898.43	2042.77
Sign RSA-1024	0.000 007	1.88	0.35
Sign RSA-2048	0.000 022	6.22	1.22
Sign RSA-4096	0.000 093	26.73	4.68
Verify RSA-1024	0.000 001	0.34	0.10
Verify RSA-2048	0.000 002	0.50	0.08
Verify RSA-4096	0.000 003	0.75	0.10
Gen EC-224	0.000 004	1.11	0.06
Gen EC-256	0.000 002	0.51	0.05
Gen EC-384	0.000 009	2.65	0.07
Gen EC-521	0.000 018	5.25	0.27
Sign EC-224	0.000 005	1.43	0.15
Sign EC-256	0.000 003	0.83	0.18
Sign EC-384	0.000 011	3.24	0.16
Sign EC-521	0.000 022	6.27	0.10
Verify EC-224	0.000 005	1.52	0.05
Verify EC-256	0.000 005	1.54	0.08
Verify EC-384	0.000 012	3.43	0.07
Verify EC-521	0.000 023	6.74	0.15
Sphinx (1x)	0.000 034	9.66	0.31
Sphinx (10x)	0.000 300	86.34	8.07
Sphinx (100x)	0.002 925	842.44	11.40

Table 3.2: Average energy consumption of different asymmetric cryptography operations. This table includes more results compared to the one in the paper.

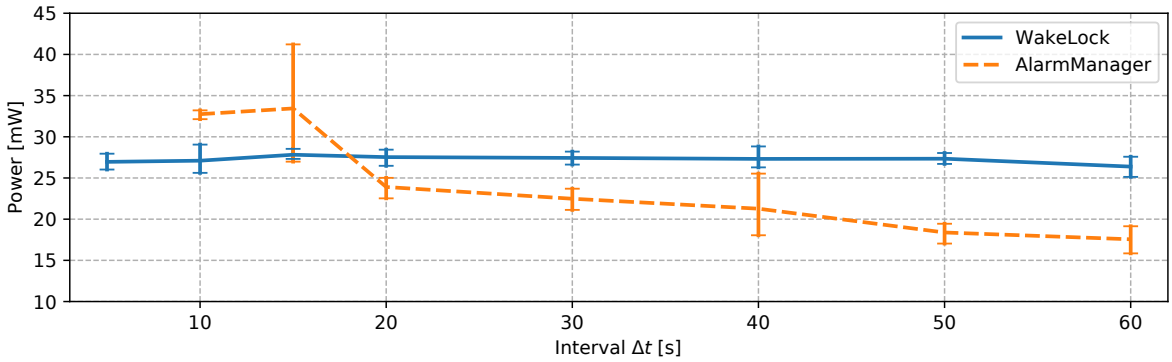

Figure 3.4: Average power consumption when using the *ForegroundService with WakeLock* and *AlarmManager* strategies.

currently communicating or not. We focus on the two main strategies for Android: *ForegroundService with WakeLock* and *AlarmManager*. Devices running iOS have similar, albeit more restrictive, mechanisms [11].

For our purposes we discuss these mechanisms as they were intended by the operating system design. Many smartphone vendors make modifications and add restrictions to achieve higher battery life which can interfere with the proper operation of apps [97]. We verified that the device we use in our studies follows the specified behaviours.

The *ForegroundService with WakeLock* strategy holds a lock that prevents the phone from entering full idle mode. As this affects background power consumption, the operating system requires the app to show a notification to the user by running as a ForegroundService. This approach provides the greatest flexibility and accuracy, as pauses can be implemented as simple `Thread#sleep` calls. Typically, this mode of operation is used by music apps and GPS navigation. However, even without active computation, the WakeLock causes higher energy consumption than regular idle mode.

The *AlarmManager* strategy allows the phone to go into full idle mode and registers the intended execution with an alarm service. The alarm service then wakes up the device and starts the respective application with a prepared Intent. While this allows the phone to enter full idle mode between executions, every wake-up comes with overhead as the system restores state and (re-)delivers the invocation arguments to our application. The AlarmManager's intended use are events that happen only a few times per hour. We found that the lowest reliable inter-execution pause is 10 seconds and that the execution times are imprecise. We compensate for the latter by measuring the time between the scheduled and the actual execution, and then applying this delta when scheduling the next alarm. However, this compensation is imperfect as the effective delays vary.

We use Δt to denote the duration between the start times of two consecutive operations. From the documentation we expect that using WakeLocks is more efficient for smaller Δt. While keeping the CPU awake generally raises power consumption, WakeLocks do not cause extra overhead for every execution. On the other hand, the AlarmManager should perform better for larger Δt as it allows the phone to reduce power consumption during longer pauses.

In our experiment we trigger regular execution for various intervals Δt with both the WakeLock and the AlarmManager approach. Each experiment runs for 5 minutes and is repeated 5 times. The results are shown in Figure 3.4. As expected, the WakeLock approach incurs constant cost ($\approx 27\,\mathrm{mW}$) regardless of the chosen interval. The AlarmManager approach is more efficient for $\Delta t > 20\,\mathrm{s}$. Generally, it does not differ more than $\approx 10\,\mathrm{mW}$ from the WakeLock approach in either direction. Figure 3.5 shows annotated power traces for both approaches. These highlight for the WakeLock approach the additional power consumption that happens before and after execution of the payload code.

3.3.3 Radio transmission

This micro study evaluates radio operations such as sending and receiving data via WiFi and mobile network to measure the impact of connection type, payload size, and schedule. The results from this study guide the parameterisation and evaluation of a mix network client with cover traffic in Section 3.4.3.

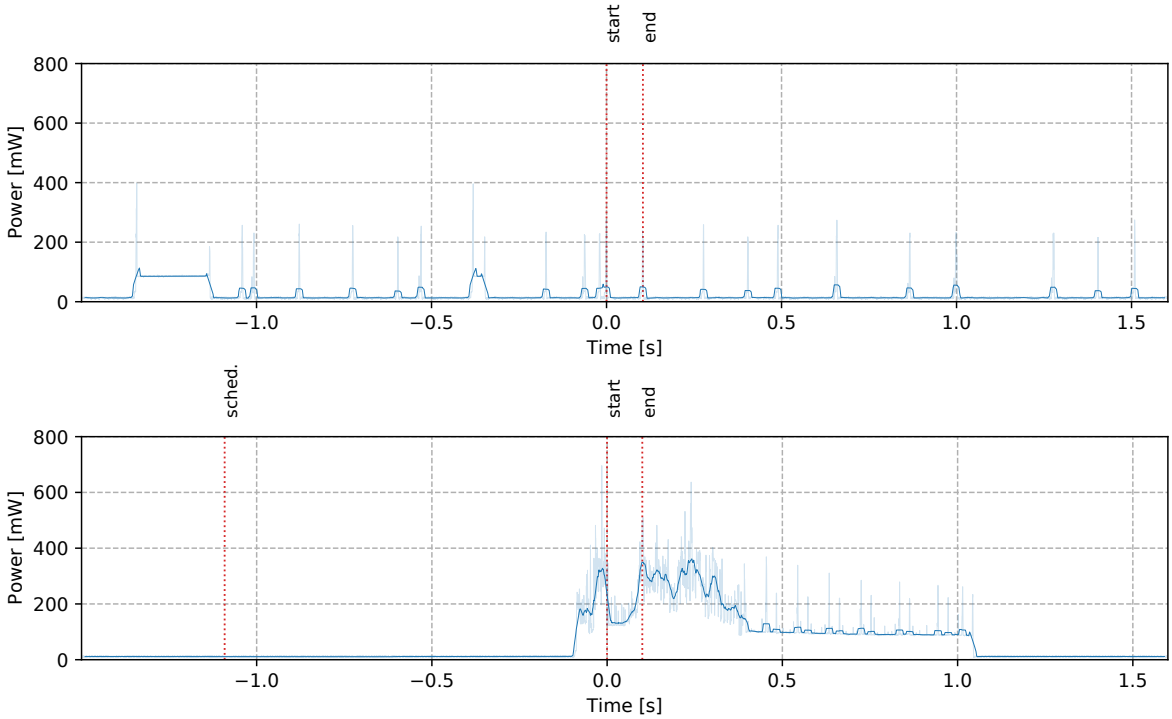

Figure 3.5: Execution of a test operation (100 ms sleep) using WakeLock (above) and AlarmManager (below). The WakeLock keeps the entire system awake which shows in a slightly higher idle power consumption and more frequent wake-ups for small operations in other apps and services. The AlarmManager allows to enter an uninterrupted low-power idle mode, but involves additional overhead before and after the payload code. One can also see the difference between the scheduled execution time marker SCHED. and the actual execution begin START. The light blue line shows the raw measurements while the dark line is a rolling average.

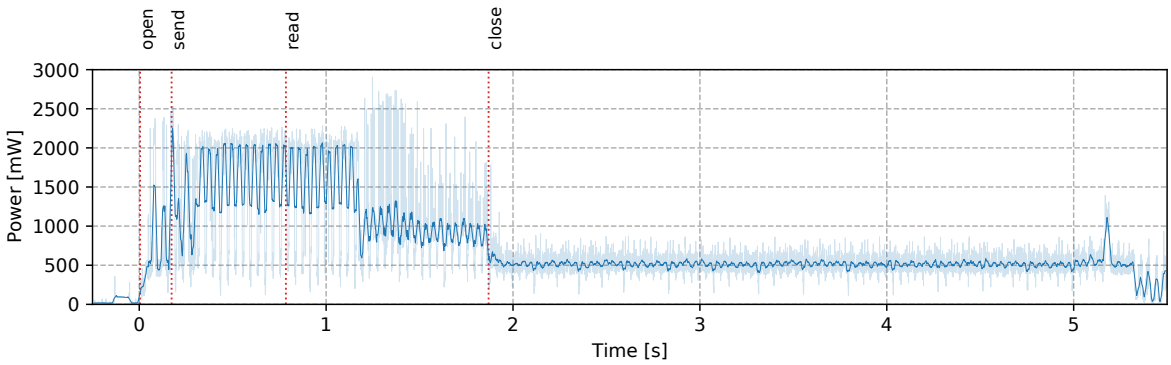

Figure 3.6: Power trace of sending and receiving 1 MiB over TCP on a 4G mobile network. When comparing with other the power traces in this chapter note the larger range on the Y-Axis. The power consumption before the SEND marker includes the transfer from idle to connected mode. The power consumption after the CLOSE marker is an effect of the tail latency where the connection is kept alive in anticipation of further incoming or outgoing packets. The light blue line shows the raw measurements while the dark line is a rolling average.

As mentioned in Section 3.1.2, radio communication, and in particular mobile networks, are complex due to their internal state machine and tail latencies. This means that their behaviour applies globally to the smartphone and not for each app independently. Hence, one cannot consider transfers individually, but need to include the effects of previous and concurrent transfers. Figure 3.6 shows an annotated measurement of a TCP data transfer via 4G. While WiFi has negligible tail latencies, we found that it is more susceptible to noise from other devices, as the smartphone is woken up regularly to process incoming broadcast packets.

In this study the smartphone connects to a 4G mobile network using a prepaid SIM card from the UK provider GiffGaff which uses the O2 network. Results for 3G are not included since 4G is widely supported and more popular. Before each experiment we verified that the mobile phone has good reception. For the WiFi measurements, we setup an access point that is secured using WPA2-PSK which is a typical setup for many consumer routers.

We execute data transfers as individual operations using the EnergyRunner app. The app uses the WakeLock method to control the scheduling of background operations. We chose the WakeLock over AlarmManager, as its average power consumption is constant regardless of the chosen interval length. This allows us to subtract the higher idle power consumption from the measured data in order to separated out the costs for radio communication. The network operations connect to a custom server that we run on a virtual machine which is located in a nearby city (ping < 100 ms). In our protocol the message consists of a 16 byte secret token (to avoid abuse by third-parties), a 4 byte client length field, and a 4 byte server length field. The server continues reading until it consumed all data specified by the client length field and then responds with data as specified by the server length field. Both are always set to the same size.

The test configurations are run for three protocols: TCP, TCP (keep-alive), and UDP. In *TCP* mode a new connection is established for each individual transfer. In *TCP (keep-alive)* mode the app maintains a global socket connection that is used for subsequent transfers. In *UDP* mode

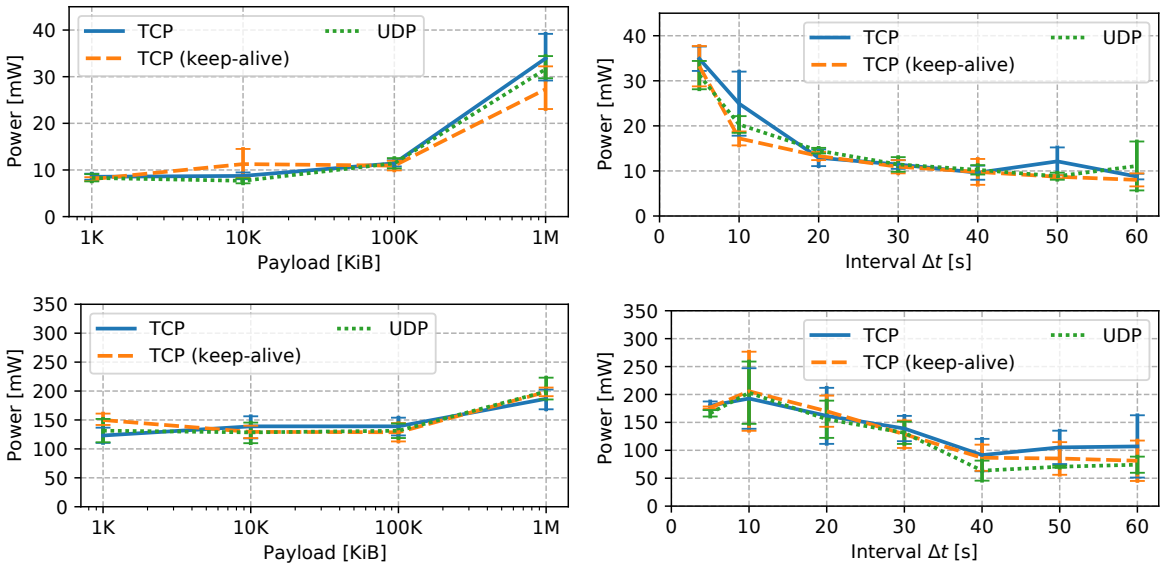

Figure 3.7: Right: average power consumption when sending and receiving 100 KiB with increasing interval times over WiFi (top) and 4G (bottom). Left: average power consumption when sending and receiving increasing payload sizes at a fixed interval of $\Delta t = 30$ seconds.

individual packets with a payload size of 1024 bytes are sent. The latter can reduce the total number of round-trips by avoiding the handshakes for each separate transfer.

We evaluate both different intervals and different payload lengths. For the former we increase the interval from 5 seconds to 60 seconds with a fixed payload size of 100 KiB. The different payload lengths range from 1 KiB to 1 MiB with a fixed interval of 30 seconds. Each configuration is tested via 4G and WiFi for 5 minutes each. We repeat all experiments 5 times and randomise their order to account for noise.

Our results for 4G and WiFi are summarised in Figure 3.7. Generally, for 4G with increasing interval times, the average power consumption drops. Notably, there is an exception for $\Delta t \approx 10$ s. With this configuration the device reconnects right after the tail latency has expired, hence maximising power consumption. We found that payload sizes up to 100 KiB have little impact on the average power consumption. We found that WiFi has much lower power consumption compared to 4G. Without the connection establishment and tail latency, the effect of increasing intervals shows more directly. Similarly, the payload size has a more direct impact.

3.4 Macro studies of protocols

The first two macro studies investigate the impact of VPNs and a mobile Tor client where we are interested in the effect on battery life for different configuration and scenarios. The third macro study measures a mix network with cover traffic. For this we implement a mobile client for a Loopix-like protocol and evaluate different sets of parameters.

In these macro studies we primarily look at average energy consumption across idle scenarios, where the protocol implementation runs in the background without any user interaction, and active

scenarios, where we simulate user interaction. The simulated user interactions follow simple schedules, such as opening a website every minute, to enable easy comparison and reproducibility. In this context, the results that we present in Figure 3.8 are overall average power consumption values (measured in mW) assuming that these scenarios run in isolation for a long time. As such, the idle scenarios give a lower bound on the energy consumption for the given implementation and connectivity which we use for our feasibility discussion in Section 3.4.5. This also means that the presented numbers do not allow estimating the costs of individual operations, e.g. loading a website. For more realistic estimates of average power consumption throughout a typical day, one can compute a weighted average of the measured power consumption accounting for the distribution of active usage times and availability of 4G and WiFi.

3.4.1 VPN

This macro study examines the question of how much VPN clients impact battery life. For this we examine two popular, commercial services ExpressVPN (version 10.89) and Proton VPN (version 4.6.12) which are marketed for personal use on mobile devices. We randomly blind them as VPN_A and VPN_B. We created paid-for subscription accounts with both providers and then installed their most-recent apps on our test device. Where possible we use the default configuration and connect to an endpoint in New York. For Proton VPN we changed the protocol from *Wireguard UDP* to *Wireguard TCP*, as the former was less reliable in our setup.

Our measurement setup captures the total device power consumption while the VPN service is running. This ensures that we include all direct and indirect effects that affect battery life. We run an *idle scenario* where no user action is simulated and a *web browsing scenario* where an instrumented web browser loads the start page of the New York Times every 60 seconds using the `AlarmManager` scheduling strategy. The news site was chosen as it has a typical size (≈ 3 MiB) and requires multiple connections to different domains. The AlarmManager strategy was chosen to allow return to idle. Our implementation holds a temporary `WakeLock` while loading the page until the Chromium-based `WebView` signals completion. The study covers all combinations of activities (idle and web browsing), radio (4G and WiFi), and network configuration (direct, VPN_A, and VPN_B). Each run lasts 10 minutes and similarly to the previous studies we deactivate all other apps and keep the screen off.

The results are summarised in Figure 3.8. First we look at the baseline measurements without any VPN service (direct). The idle scenario shows that smartphones can achieve very low power operations when there is no background activity. Notably, the active loading of the website has a strong effect for 4G, but not on WiFi. This is likely due to the aforementioned connection state changes and tail latency. Both VPN providers have similar overhead. Where they differ we pick the lower power consumption, as we are interested in a competitive baseline to compare Tor and Loopix against. During the idle scenario, a VPN adds around 40 mW or 0.5 percentage points per hour (pp/h) on 4G. This is less pronounced on WiFi due to negligible tail latencies: 8 mW (0.1 pp/h). For web browsing, the overhead increases to 80 mW (1.0 pp/h) on 4G and 20 mW (0.2 pp/h) on WiFi.

We expected a smaller overhead from the VPN apps—especially when the device is idle. Inspection of the acquired power traces shows that while the clients are running, the device does not reach low-power mode, but instead regularly wakes up to send and receive data. This is surprising, as the tested Tor client (Section 3.4.2) is able to run with slightly less power overhead. We cross-checked

Figure 3.8: Relative power usage of the tested VPN clients (left) and Tor modes (right) compared to direct network connections. The measurements for *direct* were only done once for each scenario excluding the `.onion` case where it does not apply.

the results with an OpenVPN client and a self-hosted server which led to similar measurements. We suggest future work to explore the VPN client implementation and configuration space in full detail.

3.4.2 Tor

This section examines the question of how much Tor impacts battery life of mobile devices. For this we examine Orbot [128], which uses the official Tor client under the hood. We downloaded version `16.6.0-RC-4` from the official repository and added an option to force off connection padding (see below). The app is compiled in release mode and then installed on the test device. For our evaluation we run Orbot in VPN mode which emulates a VPN client on the device and ensures that all communication is routed through Tor. This is different to the behaviour of the Tor Browser which only protects the communication of the bundled browser. We use the same measurement setup as for the VPN services in Section 3.4.1.

Tor supports rudimentary cover traffic through connection and circuit padding [107]. Connection padding affects the connection to the first hop (the Guard node). When active each payload packet causes both ends (i.e. the client and the Guard node) to sample a timeout between 1.5 and 9.5 seconds. If no other payload packet is sent before the timeout expires at either end, a single padding cell packet is sent and the timeouts are reset. To reduce the overall overhead, the client may negotiate a reduced mode where the Guard node does not send padding cells and the client samples a timeout between 9.0 and 14.0 seconds. A client may also completely disable connection padding. We test all three variations (full, reduced, disabled) of connection padding as they have a large impact on radio communication. Circuit padding is independent of connection padding and aims to obfuscate the setup phase of onion circuits. The client will send obfuscated packet sequences so that different

circuit types result in similar looking packet exchanges. We verified that circuit padding has negligible impact regardless of the chosen connection padding. Hence, we do not include it as a parameter for our experiments.

As in Section 3.4.1, this study covers run all combinations of activities (idle and web browsing), radio (4G and WiFi), and network configuration (direct and the various Tor connection padding modes). We exclude the initial connection phase to the Tor network as this a one-time cost. The Tor Consensus document ($\approx$600 KiB compressed) needs to be updated every three hours. We argue that it is negligibly small compared to other web traffic.

The results are summarised in Figure 3.8. For discussion of the *direct* connection without any services running, see Section 3.4.1. Using Tor without any padding has little effect on the idle scenarios. From our measurements we calculate an average power consumption of 59 mW or 0.7 percentage points (pp) per hour (WiFi: 0.3 pp/h). For 4G the power consumption increases by <30 mW compared to not using Tor whereas the impact is negligible on WiFi. However, for the web browsing scenario, enabling Tor increases the average costs by around 150 mW (WiFi: 60 mW). This is mostly due the lower bandwidth and high latency which both increase the time website's loading time and hence the time the radio module is active.

Enabling full connection padding in Tor has a significant effect on battery life. On 4G the average idle power consumption increases to 2.9 pp/h (WiFi: 1.1 pp/h). The relative factor for 4G (7×) is higher than that of WiFi (3.5×). The difference can be explained by the costs on 4G for the connection state changes. With reduced padding the average power costs are lower than the average increase of the interval would suggest. We found that this is because the longer gaps allow the entire system to enter a low-power state from which it often does not wake up again itself. Examination of the source code showed that Orbot itself does not use `WakeLocks` or `AlarmManager`—so it does not directly prevent the system from entering idle mode. Note that our web browsing scenario wakes up the entire system including Tor which then stays active for a while. This explains why the impact of Tor is larger in the Web scenarios.

Tor can also be used to access onion services by using their `.onion` address. Through a directory look-up and an anonymous rendezvous point, the client establishes an anonymous connection with the service without learning its server IP or location. The New York Times offers such an onion service[2]. Overall we observe a slightly increased power consumption compared to opening the regular homepage via Tor, which can be explained by the multi-round connection setup and the longer path to connect to an onion service.

3.4.3 Mix network

Finally, we evaluate mix networks operations based on the Loopix [110] design. We chose it because of its integrated support for devices that are temporarily offline and note that it is successfully deployed in the commercial mix network Nym [49]. In Loopix the provider node manages access to the network and maintains an inbox of messages for the client. This allows the client to retrieve messages later when it was offline. Hence, all connections of the client go through one provider node which simplifies our setup as we can ignore the rest of the network. We recall from Section 2.1.4 that Loopix clients

[2] `https://www.nytimesn7cgmftshazwhfgzm37qxb44r64ytbb2dj3x62d2lljsciiyd.onion/`

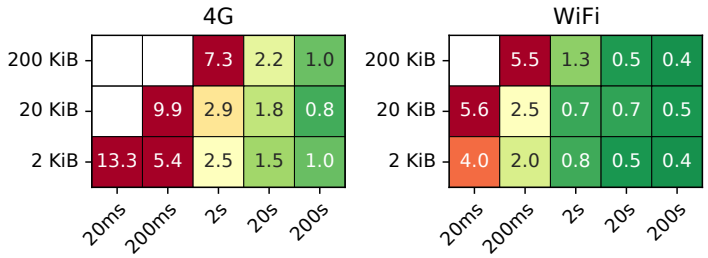

Figure 3.9: Measured energy consumption of a Loopix-style anonymity network for given mean message intervals ($1/\lambda$) and packet size (p). The colour scale ranges from ● 0 pp/h to ● 5 pp/h. Blank squares indicate excluded configurations.

use traffic shaping to hide whether actual communication is happening or not. This means that in in this evaluation we can ignore the presence of actual payload messages, as they neither influence the number of cryptographic operations nor have an influence on the sent traffic schedule and bandwidth. Hence, all cryptographic operations and radio transmission solely depend on the message rate λ and message size p.

Our evaluation uses the *WakeLock* approach for background scheduling. For each round we execute Sphinx once for the outgoing packet and transmit a message of size p (the encrypted packet) via UDP to a provider node. The provider replies with p bytes (the inbox content). Afterwards we draw a pause from the exponential distribution with parameter λ and wait for the remaining time until the start of the next operation. Each scenario is executed for 20 minutes on both 4G and WiFi. For our parameter choice of λ and p we first identify practical limits. On our test device a Sphinx operation and sending a UDP packet takes around 20 ms. Using a speed test we measured that our 4G connection provides up to 2 Mbits/s upload (WiFi: 50 Mbit/s) and exclude configurations exceeding these limits. We pick 2 KiB as our smallest packet size. This is also the size used by Nym. For $\lambda = 20$ ms the theoretical throughput is 100 KiB/s (40 % of the available bandwidth). In the Sphinx implementation each packet has a constant overhead of 365 bytes resulting in an application level goodput of 82 KiB/s. Larger packet sizes improve the goodput to throughput ratio.

The results are shown in Figure 3.9. By using the same factors for both parameter scales, data points on diagonals (up and to the right) share the same bandwidth. For $\frac{1}{\lambda} = 20$ ms on 4G both CPU and network are consistently active and deplete the entire battery in 7.5 hours. The results also show that for the same bandwidth, larger intervals lead to significant energy savings. For example, with 10 KiB/s of bandwidth, the energy used decreases from 5.4 pp/h ($p = 2$ KiB, $\frac{1}{\lambda}$=200 ms) to 2.2 pp/h ($p = 200$ KiB, $\frac{1}{\lambda} = 20$ s) in the 4G case and from 2.0 pp/h to 0.5 pp/h for the same parameters in the WiFi case. However, this comes at the cost of increased latency.

3.4.4 Daily driver

The results from the macro studies suggest that some configurations can be run continuously during normal daily usage. To validate these conclusions, a "Daily Driver" scenario is used that mimicks typical smartphone usage while running anonymity networks. Assuming an informed user, who prefers the most energy-efficient configuration, we choose VPN_A and *Tor without padding*. For the mix network we pick two parameters from the Loopix paper as $Loopix_{fast}$ ($\frac{1}{\lambda} = 2$ s) and $Loopix_{slow}$

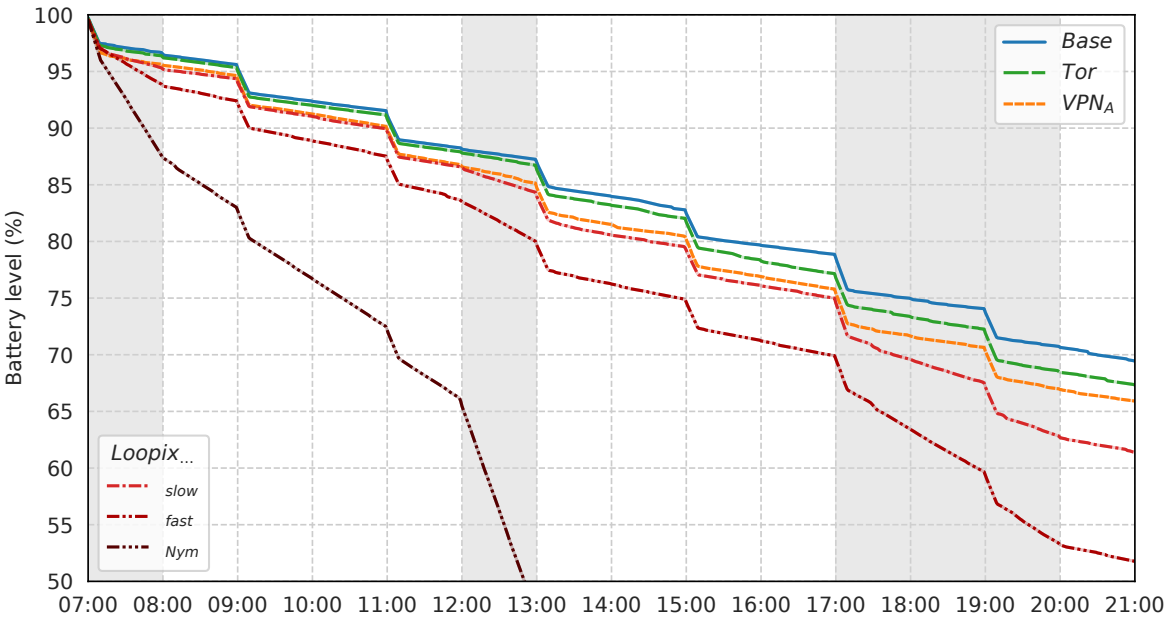

Figure 3.10: Measured energy consumption as estimated battery levels during our 14 hour daily driver scenario. A white background indicates connection via WiFi (grey: 4G).

($\frac{1}{\lambda} = 20\,\text{s}$). As the paper does not mention concrete packet sizes, we choose $p = 20\,\text{KiB}$ which can handle long text messages. We also include the default Nym configuration ($\frac{1}{\lambda} = 20\,\text{ms}$, $p = 2\,\text{KiB}$) as $Loopix_{Nym}$. All are compared to a *Base* configuration without any anonymity network client.

The experiments are performed continuously for 14 hours simulating a typical day from 7 am to 9 pm. The smartphone has been reset to factory state, fully charged, and the following applications are installed: Google Mail, Signal, YouTube. The device is connected to WiFi most of the day, except for 7–8 am, 12–1 pm, and 5–8 pm when it then connects via 4G. We automatically send an email to the Gmail app every hour and a text message using Signal every 15 minutes. Both wake up the device to show a notification. In addition, we play a 10-minute YouTube video every two hours starting at 7am to simulate longer Internet sessions with multiple HTTPS requests. The display brightness is set to 75%.

The results are shown in Figure 3.10. Unsurprisingly, most energy is consumed when the screen is active (around 1 000 mW). Push-notifications appear as small drops every 15 minutes. Overall, the findings agree with our previous experiments. In this specific scenario the overall energy overhead for *Tor* and VPN_A are 0.2 pp/h and 0.3 pp/h respectively. Hence, both appear practical. The change in the gradients for Loopix make the power consumption differences between WiFi and 4G clear (e.g. $Loopix_{fast}$ around 12 noon). The high-latency $Loopix_{slow}$ configuration has an average overhead of 0.6 pp/h ($Loopix_{fast}$: 1.3 pp/h) which is almost practical when there is WiFi for most of the day. The $Loopix_{Nym}$ parameters drain the battery completely after 12 hours.

3.4.5 Discussion on feasibility

We discuss the feasibility of VPNs, Tor, and mix networks with cover traffic based on our assumptions from Section 3.1: smartphones are used without charging for a 12-hour period and using less than an extra 5 percentage points of battery during this time is acceptable. The results can be easily adjusted if required. We benchmark the anonymity networks against an idle phone with active network connection which consumes around 0.4 percentage points per hour.

We evaluate whether it is feasible to run Tor without padding in the background by considering the idle scenario on 4G. This gives us a lower bound of the actual costs. Note that active usage and background communication will experience additional overhead due to higher latency and lower bandwidth. For a 12-hour usage period running Tor without connection padding on 4G requires an extra 3.6 pp (WiFi: <0.1 pp). Similar for the other numbers. This is less overhead than from the tested VPN clients (Section 3.4.1). We conclude that Tor without connection padding is feasible on modern smartphones and can be run continuously without large drawbacks. However, full connection padding has a large impact. This configuration increases energy costs by 30.0 pp on 4G and 9.6 pp on WiFi which is no longer practical.

We evaluate the Loopix-style mix network using the same methodology. In contrast to Tor, active usage does not increase the overall energy overhead of the mix network client as the scheduling and size of packets does not change. We consider a packet size of 20 KiB for both a medium-latency ($\frac{1}{\lambda} = 2\,\text{s}$) and a high-latency ($\frac{1}{\lambda} = 20\,\text{s}$) configuration. A 2 second latency would be acceptable for text-based chats and our evaluation shows that it requires an additional 30.0 pp on 4G for a 12-hour period. The high-latency configuration fares better with 16.8 pp for 12 hours. While the numbers are acceptable on WiFi (both 4.8 pp), using local networks can not fully compensate for the high energy costs of 4G networks, as we expect smartphone users to use their devices in many different locations. Only very high-latency parameters, e.g. $\frac{1}{\lambda} = 200\,\text{s}$, would have a small enough overhead (4G: 6 pp, WiFi: 2.4 pp) to be considered almost practical.

3.5 Limitations and threats to validity

The presented evaluation uses only one smartphone which limits how well the results generalise. We chose the Motorola phone as our test device because of the easy battery access so that others can easily replicate our setup and results. As it runs a mostly unmodified version of Android it reflects the intended behaviour of the operating system. We verified this by running the Don't Kill My App [97] benchmark which schedules and later verifies various background operations. Our test device received a perfect score. Other vendors add custom battery saving techniques which can introduce bugs or restrict background services [97]. This applies also to iOS devices which are more restrictive on background activities. Therefore, our results may not directly generalise to other devices in terms of functionality and power consumption. Running background cover traffic over long periods of time might even be prevented by the operating system on some platforms without explicit intervention by users. Similarly, differences in modem hardware and configuration affect the obtained power measurements. However, we believe that results for devices with similar functionality are comparable, with newer devices generally being more energy efficient.

We chose GiffGaff as the mobile network provider as they offer pre-paid SIM cards. While it appears representative to us based on our observations, the results do not necessarily translate to other providers. For instance, the duration of tail-latency demotion is a parameter that is set by the network provider and can change over time. The GiffGaff configuration has a fairly short tail-latency which appears favourable for regular small messages. Therefore, the results for mix networks can be seen as a lower-bound estimate and we might expect it to be higher for other providers.

For our macro studies we use a public website hosted by a third-party. Updates to the website will inadvertently change the obtained absolute measurements. Likewise, mobile network conditions are not perfectly reproducible. For example, upgrades by the provider and testing in different locations will lead to slightly different numbers. As a mitigation, we always (re-)ran all VPN and Tor experiments together at the same location to ensure comparability of the results.

3.6 Summary

In this chapter we first developed an easy-to-build hardware setup that installs a power sensor between a smartphone and its battery and logs the data to a PC. We can use this directly to measure the average power consumption of long-running services. In addition, we can record timestamps before and after the execution of short operations on the device to later synchronise the data and calculate the energy consumption of individual operations precisely. Compared to previous work, which uses models or modifies apps or the operating system, our setup gathers ground-truth data without modifications to software or hardware. This makes sure that all side-effects, such as two apps accessing the radio at the same time, are always correctly reflected in the data.

Using our setup we showed that performing cryptographic operations on smartphones, especially when using modern algorithms, is cheap. However, radio transmissions can be expensive. Interestingly, it is primarily the timing of the send and receive operations—not the total amount of data being transferred—that dominates battery usage. With this background it is encouraging to see that the most popular anonymity technologies (VPN and Tor) can be run continuously during the entire day in the background with minimal impact. However, designs with stronger anonymity guarantees such as mix network designs like Loopix do not work well on smartphones for low-latency configurations.

When using these existing mix network designs it is therefore important to keep the sending rate low and opt for larger message sizes where we need higher bandwidth. However, this of course limits how many messages one can send in bursts, e.g. to inform many different recipients in a group chat. These messages would be stuck for a while in the outgoing payload queue as each one would have to wait for its own independent sending event. We address this particular scenario in the next chapter where we present a protocol for efficient group-multicast in mix networks.

Chapter 4

Low-latency group communication in mix networks with unreliable connectivity

Mix networks typically only consider one-to-one (unicast) communication where each message is sent to exactly one recipient. As such, they have no built-in mechanism for multicast where a sender can efficiently share a message with all members of a group. In particular, naïvely implementing multicast through sending many individual messages to the group members results in significant overhead in terms of latency and throughput, typically exceeding the latency required to provide good user experience. This effect is aggravated on mix networks due to the (cover traffic) sending schedule that limits how quickly messages are picked-up from the payload queue.

The need for sending messages to many users arises not only in group chats, but also when collaboratively editing documents in a decentralised setting where no central server can distribute messages. In this chapter we explore the limitations of naïve multicast implementations and find that they are not suitable beyond very small groups. Therefore, we present Rollercoaster, an efficient group multicast scheme takes the particular latency and throughput constraints of mix networks into account.

In Rollercoaster, all group members help distributing a messages so that load is more evenly distributed. This results in a drastic reduction of latency and overall improves the ratio of payload to cover traffic, hence improving efficiency of the network. Most importantly, Rollercoaster acts as a layer on top of Loopix without requiring any modifications. To account for mobile devices, and in particular intermittent connectivity, we also present a fault-tolerant version of Rollercoaster that allows the group to work around unreliable participants. As an optional improvement we present a modification of the Sphinx protocol where individual messages can be addressed to two or more recipients. This modification is called MultiSphinx and can further improve efficiency and latency. While MultiSphinx requires modifications to the underlying network, it is designed to share the same anonymity set with regular messages.

This chapter is based on our paper "Rollercoaster: An Efficient Group-Multicast Scheme for Mix Networks" [68]. For this chapter I adapted the text, figures, and plots from the ROLLERCOASTER paper to fit with the dissertation. I led the design of the protocol, implemented the prototype, conducted all experiments, performed the data analysis, did the security analysis, and was the primary author of the text. Martin and Alastair contributed towards the development of the ideas and their presentation.

4.1 Group communication

A *multicast* protocol delivers a single message to multiple recipients. Broadly speaking, there are two common approaches for implementing multicast: by sending each message individually to each recipient over unicast, or by relying on the underlying network to make copies of a message that are delivered to multiple recipients. IP multicast [48] is an example of the latter approach, which avoids having to send the same message multiple times over the same link. There are many existing multicast protocols [64, 111, 150]. However, to our knowledge, none of these existing protocols accounts for the special challenges of mix networks where clients' sending of messages is artificially rate-limited as outgoing packages must not leak communication patterns. Our implementation required careful adaption and tuning to the particularities of Loopix.

For this chapter we are interested in *group multicast*, a type of multicast protocol in which there is a pre-defined, non-hierarchical group of users U. We call the initial sender *source* s and all others the intended recipients $U_{\text{recv}} = U \setminus s$. At any time any member of the group might send a message to all other group members. We also assume that the group membership is fixed and known to all members; we leave the problems of group formation and adding or removing group members for future work that we discuss in the conclusion chapter.

4.1.1 Collaborative editing and local-first software

Group messaging and collaboration can share the same underlying infrastructure [79]. For this we consider forms of collaboration in which a file or conversation thread is shared by a group of collaborators, and any update to it needs to be shared with all group members. In collaborative editing applications individual update messages are usually small and frequent [1]. Hence, such apps require an efficient, reliable, and timely method of sending messages to all members of a group.

In this context user studies highlighted the negative implications of high network delays in collaborative editing. One previous study [71] asked a group of participants to transcribe audio lectures using collaborative text editing software. The researchers investigated the effect of communication latency by repeating the experiment multiple times and varying artificial delay on all communication between participants. A delay of 10 seconds or more had a significant impact in their study, with an increase of error rates and content redundancy by more than 50%. We therefore set our target for group multicast latency at 10 seconds for group sizes of up to 100 people. The group size is motivated by the active editor limit of Google Docs (100 users) and Microsoft Sharepoint (99 users). We further require the latency to grow sub-linearly with the size of the group, allowing effective collaboration in large groups. In many multi-user applications, a large fraction of the data is generated by a small

fraction of the users—a trend that is known as *participation inequality* [98]. Rollercoaster fares well in a system with such a distribution of activity.

Group multicast protocols should also come with offline support, which is required since mobile devices do not always have connectivity. As in the Loopix design, provider nodes in Rollercoaster continue to store messages on behalf of the user until the user is next online and able to download them. For this we introduce a Rollercoaster variant that is fault-tolerant (Section 4.3.2) but achieves the same performance of Rollercoaster if all users are online.

As we discuss in Chapter 3, on mobile devices the frequency of sending network packets has a large impact on energy efficiency, as every packet can trigger state promotions that come with additional energy costs. Therefore, sending few but large messages with long pauses between packets is advantageous for battery life on mobile devices, even if the total volume of data transmitted is the same. On the other hand, smaller and more frequent messages lead to lower latency. Hence, we optimise Rollercoaster to work well when running over a mix network with infrequent messages.

4.1.2 Threat model

As in Loopix, we assume a global active adversary who can observe all traffic, manipulate traffic to remove messages and insert new ones, as well as corrupt a subset of mix nodes and providers. Sending a message to a Rollercoaster user requires that the sender knows both the addresses and public keys for their provider, the recipient, the recipient's provider, and the mix nodes.

Rollercoaster provides message confidentiality and integrity as well as the same strong metadata privacy guarantees as Loopix, including *sender–recipient unlinkability* (preventing an adversary from deducing which users are communicating with each other) and *sender–recipient online unobservability* (preventing an adversary from deducing which users are currently participating in any communication). In addition we provide *membership unobservability* which prevents anyone outside the group from determining group membership or group size. We assume a group is composed of trusted members and therefore we do not provide unlinkability or unobservability guarantees against an attacker who compromises or colludes with group members. The goal of the attacker is to break the confidentiality, integrity, or metadata privacy guarantees. Our scheme supports efficient communication for group sizes of two or more and therefore we handle pairwise and group communication in the same way. An attacker cannot distinguish between two-party communication and communication in a larger group.

4.2 Naïve approaches to multicast

In this section we discuss the reasons for why message delays occur in Loopix and capture them in analytical formulas. We then explore two simple approaches to implementing multicast on Loopix, and explain why they are not suitable, before introducing Rollercoaster in Section 4.3.

We recall from Section 2.1.4 of the background chapter the latency of a single message in the Loopix network. In particular, let $l = 3$ be the number of layers, n be the number of previous messages in the send queue, λ_p be the send queue rate, μ be the per-hop delay, and δ_{pull} the frequency for checking the inbox. Then the expected average delay for sending a single message measured from

when it is scheduled at the sender to when it is received at the recipient is:

$$\text{mean}(d_{\text{msg}}) = \frac{n+1}{\lambda_p} + \frac{l+1}{\lambda_\mu} + \frac{\Delta_{\text{pull}}}{2} \tag{4.1}$$

When a source s wants to send a payload to a group by multicast, we define the *multicast latency* D to be the time from the initial message sending until all of the recipients U_{recv} have received the message:

$$D = \max_{u \in U_{\text{recv}}} (T_{\text{recv},u}) - T_{\text{send},s} \tag{4.2}$$

4.2.1 Naïve sequential unicast

In the simplest implementation of multicast, the source user s sends an individual unicast message to each of the recipients $u \in U_{\text{recv}}$ in turn. While the messages can travel through the mix network in parallel, their emission rate is bounded by the payload rate λ_p of the sender.

For a recipient group of size $|U_{\text{recv}}| = m - 1$, the last message in the send queue will be behind $n = m - 2$ other messages. Further, the last message will incur the same network delay and pull delay as all other unicast messages. The average delay for the last message therefore describes the multicast latency for when performing sequential unicast:

$$D_{\text{unicast}} = \frac{m-1}{\lambda_p} + \frac{l+1}{\lambda_\mu} + \frac{\Delta_{\text{pull}}}{2} = \mathcal{O}(m) \tag{4.3}$$

The mean delay D_{unicast} therefore grows linearly with m. As we show in Section 4.4, sequential unicast is too slow for large groups with realistic choices of parameters (λ_p is typically set to less than one message per second).

Another problem with the sequential unicast approach is that the effective rate at which a user can send messages to the group is $\frac{\lambda_p}{m-1}$, as all copies of the first message need to be sent before the second multicast message can begin transmission. One might argue that this problem can be addressed by increasing the payload bandwidth by increasing the value for λ_p. However, this would require similar adjustments to the rates for drop and loop messages to preserve the network's anonymity properties. As these parameters are fixed across all users, this would lead to a proportional increase in overall bandwidth used by the network. Moreover, the factor by which we increase λ_p would be determined by the largest group size we want to support. As a result, users participating in smaller groups would face an unreasonable overhead. This inefficiency particularly applies to users who mostly receive and only rarely send messages.

4.2.2 Naïve mix-multicast

An alternative approach shifts the multicast distribution of a message to mix nodes. In this scheme, the source chooses one mix node as the *multiplication node*. This node receives a single message from the source and creates $|U_{\text{recv}}| = m - 1$ mix messages sent on to the other group members. A provider node would not be suitable as a multiplication node as it would learn about the group memberships of its users and their group sizes. When the multiplication node receives such a multicast message, it inserts $m - 1$ messages into its input buffer, one for each of the recipients, and processes them as

usual. This provides optimal group message latency of $D = d_{\text{msg}}$ as there is no rate limit on messages sent by a mix node, and hence no queueing delay. However, this design has significant flaws.

First, a corrupt multiplication mix node can learn the exact group size $|U| = m$, in contravention of our threat model (Section 4.1.2). This is undesirable as it may allow an attacker to make plausible claims regarding the presence or absence of communication within certain groups. Even without corrupting a node, an adversary can observe the imbalance between incoming and outgoing messages of a multiplication node.

The weakened anonymity properties could perhaps be mitigated with additional cover traffic that incorporates the same behaviour as the payload traffic. In particular, the cover traffic must model all possible group sizes. Allowing a group size of 200 requires cover traffic to multicast by factor 200 as well. However, this would significantly increase the network bandwidth requirements in the following mix layers, increasing the cost of operating the network.

Permitting message multiplication also opens up the risk of denial of service attacks: a malicious user could use the multicast feature to send large volumes of messages to an individual provider, mix node, or user, while requiring comparatively little bandwidth themselves.

Finally, supporting group multicast in a mix node requires the input message to contain $m - 1$ payloads and headers, one for each outgoing message. As all outgoing messages must travel independently of each others they must be encrypted with different keys for their respective next hops. Otherwise, all outgoing messages share the same encrypted payload. This makes it trivial for an observer to identify the recipients of this group message. The only solution is to either increase the size of *all* messages in the system or enforce a very low limit on maximum group size.

In summary, naïvely performing message multiplication on mix nodes is not a viable option. However, a viable variant of this approach is possible by fixing the multiplication factor of messages to be a small constant (e.g. $p = 2$). We discuss this design in Section 4.3.4 where we present MultiSphinx.

4.3 The Rollercoaster protocol

We propose Rollercoaster as an efficient scheme for group multicast in Loopix. Rollercoaster distributes the multicast traffic over multiple nodes, arranged in a distribution graph. This not only spreads the message transmission load more uniformly across the network, but it also improves the balance of payload and cover traffic. Rollercoaster is implemented as a layer on top of Loopix, and it does not require any modifications to the underlying Loopix protocol. However, we discuss an optional protocol modification in Section 4.3.4 that can provide additional performance.

As we saw with naïve sequential unicast, messages slowly trickle from the source into the network as the source's message sending is limited by the payload rate λ_p. However, users who have already received the message can help distribute it: after the source has sent the message to the first recipient, both of them can send it to the third and fourth recipient concurrently. Subsequently, these four nodes then can send the message to the next four recipients, and so on, forming a distribution tree with the initial source at the root.

The distribution tree for a set of users U is structured in *levels* such that each parent node has k children at each level, until all recipients are included in the tree. An example with eight recipients is

shown in Figure 4.1A. With each level the total number of users who have the message increases by a factor of $k+1$, which implies that the total number of levels is logarithmic in the group size $|U|$.

In this section we first detail the construction of ROLLERCOASTER in Section 4.3.1. As a second step, Section 4.3.2 adds fault tolerance to ensure that the scheme also works when nodes are offline. Asymptotic delay and traffic properties are analysed in Section 4.3.3. Section 4.3.4 develops the MULTISPHINX message format, which allows restricted multicast through designated mix nodes. Further optimisations to the scheme are briefly discussed in Section 4.3.5.

4.3.1 Construction

The ROLLERCOASTER scheme is built upon the concept of a *schedule*. This schedule is derived deterministically from the source s, the total set of recipients U_{recv}, and the maximum branching factor k following Algorithm 1. First, a list U of all group members is constructed with the initial source at the 0-th index. The group size $|U|$ and branching factor k lead to a total of $\lceil \log_{k+1} |U| \rceil$ levels. In the t-th level the first $(k+1)^t$ members have already received the message. All of them send the message to the next w recipients, increasing the next group of senders to $(k+1)^{t+1}$. In the 0-th level only $U[0]$ (the initial sender) sends k messages to $U[1] \dots U[k]$.

In order to associate an incoming message with the correct source node and group of all recipients, all ROLLERCOASTER payloads contain a 28 byte header as illustrated in Figure 4.2, in addition to the *Sphinx* packet header used by Loopix. Each group is identified by a 32-bit *groupid* shared by all group members. The 32-bit *nonce* identifies previously received messages, which becomes relevant with fault-tolerance (Section 4.3.2). The fields *source*, *sender*, and *role* refer to individual group members and have a 10-bit size, allowing groups with up to 1024 members. The *source* field indicates the original sender and is necessary to construct the distribution graph at the recipient. The fields *sender* and *role* are used by the fault tolerant variant in Section 4.3.2 for acknowledgement messages and to route around nodes that are offline. Field lengths can easily be increased or decreased as they do not have to be globally the same across all Loopix clients. Finally, the header contains a 128-bit *signature*[1] that is generated by the original source and covers the payload as well as all static header fields. It assures recipients that the message indeed originated from a legitimate group member and that they are not tricked by an adversary to start distributing a fake message to group members. The *sender* and *role* fields are not covered by the signature, allowing nodes that are not the original source to modify these fields without invalidating the signature.

4.3.2 Fault tolerance

The basic ROLLERCOASTER scheme of Section 4.3 fails when users are offline and cannot perform their role of forwarding messages. In this case, one or more recipients in later levels would not receive the message until their parent node returns online. The risk of this approach becomes apparent when looking at the graph in Figure 4.1B, where a single unavailable node causes message loss for its entire subtree. In principle, the responsibility for forwarding messages could be delegated to the provider

[1]The original paper [68] implied a 32-bit signature field which is clearly not sufficient. The text and the related Figure 4.2 in this chapter are updated to use a 128-bit signature field. This change does not impact our evaluation.

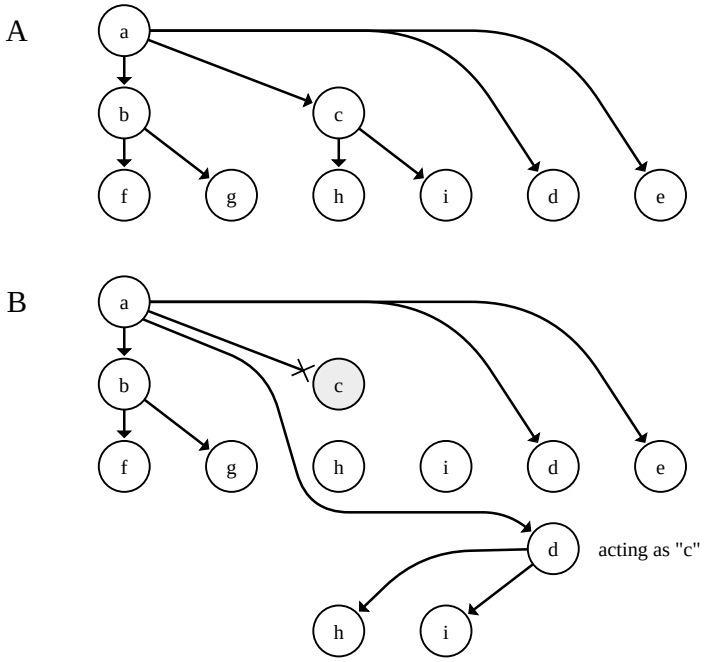

Figure 4.1: Message distribution graph for a group of size $m = 9$ and branching factor $k = 2$. *Graph A:* Expected delivery from source $s = a$. *Graph B:* The node c is offline and breaks delivery to h and i. Using the fault-tolerant variant (Section 4.3.2) the node d is assigned the role of c and delivers the payload to h and i.

Algorithm 1 The basic ROLLERCOASTER schedule algorithm for a given initial source s, list of recipients U_{recv}, and branching factor k. The *schedule* contains a list for every level with a tuple $(sender, recipient)$ for each message to be sent.

1: **procedure** GENSCHEDULE(s, U_{recv}, k)
2: $U \leftarrow [s] + U_{\text{recv}}$
3: $L \leftarrow \lceil \log_{k+1} |U| \rceil$ ▷ number of levels
4: schedule $\leftarrow [\,]$
5: **for** $t = 0$ until $L - 1$ **do**
6: $p \leftarrow (k+1)^t$ ▷ first new recipient
7: $w \leftarrow \min(k \cdot p, |U| - p)$
8: $R \leftarrow [\,]$
9: **for** $i = 0$ until $w - 1$ **do**
10: $idx_{\text{sender}} \leftarrow \lfloor \frac{i}{k} \rfloor$
11: $idx_{\text{recipient}} \leftarrow p + i$
12: $R[i] \leftarrow (U[idx_{\text{sender}}], U[idx_{\text{recipient}}])$
13: schedule$[t] \leftarrow R$
14: **return** *schedule*

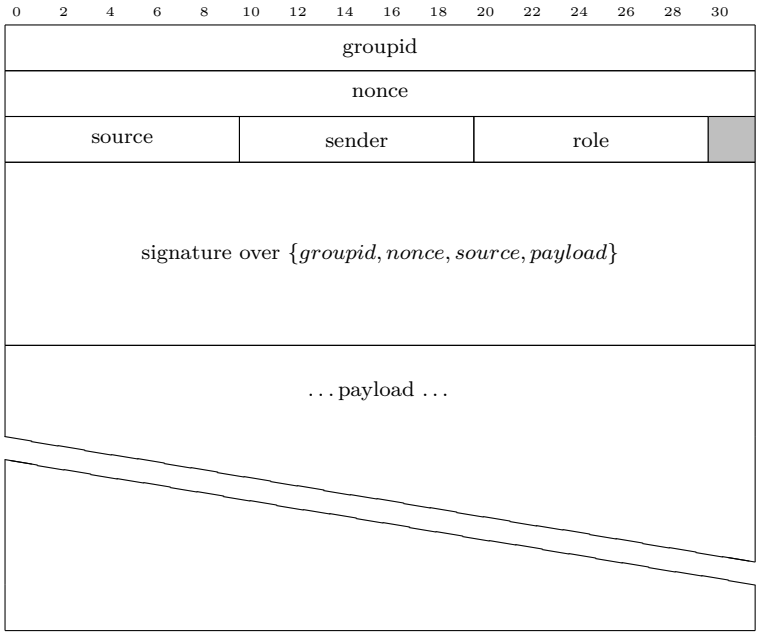

Figure 4.2: Payload header for the ROLLERCOASTER scheme containing both the fields for the minimal scheme and the fields necessary for the fault-tolerance variant and further optimisations.

nodes, which are assumed to always be online. However, we consider this approach not to be desirable as the adversary could learn about the group membership by compromising a provider.

ROLLERCOASTER with fault-tolerance achieves reliable delivery through *acknowledgement* (ACK) replies to the source and reassignment of roles. When the source sends a message it sets timeouts by which time it expects an acknowledgement from the recipient and each of its children. The individual timeouts account for the number of hops and the expected delays at each hop due to mix node delays and messages waiting in send queues. ACKs are sent through the mix network like any other unicast message. When receiving an ACK from a node, the source marks the sending node as delivered. Choosing the source as the main coordinator is reasonable as it has the strongest incentive for ensuring delivery of all messages. Loopix allows a high rate of messages received by users, so it is not a problem if one user receives a large number of ACKs.

The source responds to a timeout by sending the message to a different node. For this, each node maintains a list of most-recently-seen nodes based on received messages and chooses one from it heuristically. The source itself is part of that list as the ultimate replacement node. A replacement node is only necessary when the failing node would have forwarded the message to others, i.e. when it is not a leaf node of the distribution tree (Algorithm 9 in Appendix A.1). Independently of this and in case that the message did not reach the intended recipient due to message loss, a *retry message* is sent (with exponential back-off) to the failed node again with its own timeout.

We start the timeouts associated with a message when the underlying Loopix implementation sends the message to the provider, so that the timeouts do not need to include the sender's queueing delay. Since the sender knows the global rate parameters λ_p and λ_μ, it takes these into account when

determining timeouts. The timeout may further be adjusted based on the network configuration and application requirements.

The fault-tolerance mechanism makes use of the message fields *source*, *sender*, and *role* shown in Figure 4.2. The *source* field remains unchanged as the message is forwarded because it is required for constructing the schedule at each node. It also indicates the node to which the ACK should be sent. The *sender* field is updated when forwarding a message or sending an ACK and used by the recipient to update their list of most-recently-seen nodes. The *role* field indicates the role that the receiving node should perform, usually their natural identity. However, when a node is offline, another node might be assigned its role, i.e. its position in the distribution tree. In this case, the role field indicates the node as which the recipient should act. Retry messages to failed nodes have an empty role field, because the role has already been reassigned.

On receiving any payload message *msg*, the recipient node hands over the payload to the application and reconstructs the schedule using *msg.source*, *msg.groupid*, and *msg.nonce*. For every child node of *msg.role* in the schedule, the node enqueues a message for the respective recipient, making sure to update *msg.role*. The ACK reply is enqueued after the payload messages so that no ACK is sent if a node goes offline before forwarding a message to all of its children in the distribution tree.

ACK messages contain the *groupid*, *nonce*, *source*, and *role* fields of the original message and an updated *sender* field, which allow the recipient of the ACK (i.e. the source) to identify and cancel the corresponding timeout. The sender adds a signature covering *all* header fields to ensure that the ACK message cannot be forged. When an ACK is not received on time, the message is sent to a different node as described above.

If the connection between a user and their provider is interrupted, we rely on the fact that Loopix allows users to retrieve received messages from their inbox later. The user's software notices a loss of connection and pauses timeouts until it is able to check the inbox at the provider again.

After a long offline period, a node's inbox may contain a large backlog of messages that were received by the provider while the user was offline. When a node comes back online, it treats this backlog differently from messages received while online: for any messages received while offline, a node only delivers the payloads to the application, but it does not send ACK messages or forward messages to other nodes. Here the node avoids doing unnecessary work for messages where the timeout is likely to have already expired. Algorithm 10 in Appendix A.1 describes the behaviour of the fault-tolerant variant in detail.

Eventual Delivery and Byzantine Fault Tolerance

The fault-tolerant variant of Rollercoaster assumes that the source node acts honestly and does not disconnect permanently (but can do so intermittently). This is reasonable as the sending user has high incentive to see through the delivery of their message. We provide a summary of a proof under this assumption below and refer for its full version to Appendix A.4.

The key insight is that everyone who does not ACK the payload will eventually receive it directly from the source, and will read it from their inbox when they return online. This works even in the presence of malicious nodes that acknowledge a message without forwarding it, since the source has individual timeouts for each group member. Therefore, the source will detect when a node's children do not send ACKs.

However, the protocol can be extended to handle the case where the source node might be disconnected permanently and at least one recipient of the original message remains available. To nevertheless guarantee eventual delivery, every group member can periodically pick another group member at random and send it a hash of the message history it has seen so far (ordered in a deterministic way so that two users with the same set of messages obtain the same hash). If the recipient does not recognise the hash, the users run a reconciliation protocol [78] to exchange any messages that are known to only one of the users. Such a protocol provably guarantees that every user eventually receives every message, even if some of the users are Byzantine-faulty, provided that every user eventually exchanges hashes with every other user [78].

4.3.3 Analysis

We first analyse the expected multicast latency of ROLLERCOASTER without fault tolerance by considering the levels of the distribution tree, as illustrated in Figure 4.1. The expected multicast latency $D_{rollercoaster}$ is determined by the longest message forwarding paths $C_1, C_2, \ldots$. Each such path is defined as $C = e_0 \ldots e_{|C|-1}$ where e_i is a edge from a node on level i to a node on level $i+1$. We call these edges *one-level edges*. The number of levels of the schedule generated by Algorithm 1 is $L = \lceil \log_{k+1} |U| \rceil$ as discussed in Section 4.3. Hence, no path is longer than L. An example of a longest path is $C = (a, b)(b, g)$ in Figure 4.1. The mean message delay when traversing each edge of the graph is $\bar{d}_{\text{msg}} = \bar{d}_Q + \bar{d}_t$, where $\bar{d}_Q$ is the mean queueing delay and $\bar{d}_t = \bar{d}_p + (l+1) \cdot \bar{d}_\mu + \bar{d}_{\text{pull}}$ is the message's mean travel time through the network. Since each node sends no more than a total of k messages to the directly subsequent level, the expected queueing delay for the last message is $\bar{d}_Q = \frac{k-1}{\lambda_p}$.

However, there are also edges from a node on level i to a node on level $i+j$ where $j > 1$. One example is (a, d) in Figure 4.1. Messages from level i to level $i+1$ are sent before any messages that skip levels, and therefore any level-skipping messages may experience higher queueing delay before they are sent. Concretely, the edges from level i to level $i+j$ will incur an additional expected queueing delay of at most $(j-1) \cdot \bar{d}_Q$ compared to one-level edges. At the same time, these edges save $j-1$ hops, which would have incurred both a queueing delay $\bar{d}_Q$ and a travel time $\bar{d}_t$ each. Hence, the time saved by the reduced hop count outweighs the extra queueing delay.

Thus, the expected time for a message to be received by all nodes is determined by the longest path consisting of only one-level edges, with a queueing delay of $\bar{d}_Q = \frac{k-1}{\lambda_p}$ at each hop:

$$D_{rollercoaster} = L \cdot (\bar{d}_Q + \bar{d}_t) = \lceil \log_{k+1} m \rceil \cdot \bar{d}_{\text{msg}} \tag{4.4}$$

Hence, the group multicast latency is logarithmically dependent on the group size m and contains a multiplicative factor that equals the time to send a single message after being queued behind at most k messages.

When a node is offline, it will only be able to receive messages when it comes online and queries its inbox. In case the offline node is a forwarding node, the source will detect the lack of an ACK after the timeout expired. In this case the latency penalty for the children of the failed node is the timeout of the parent node, which is typically proportional to the expected delivery time.

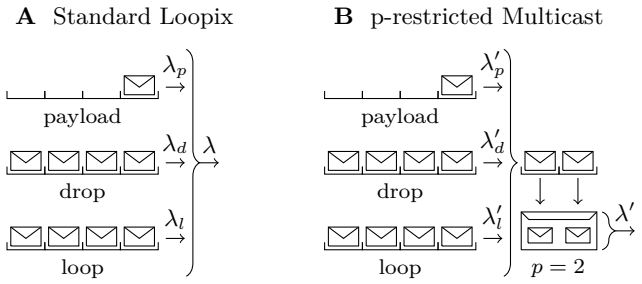

Figure 4.3: Standard Loopix (A) sends out a message if any of its Poisson processes triggers, so the rate of messages sent is $\lambda = \lambda_p + \lambda_d + \lambda_l$. In p-restricted multicast (B) these Poisson processes are still independent, but the node has an extra layer that awaits p messages, which are then wrapped into a MultiSphinx message. The sender can increase λ'_p to $p\lambda_p$ (same for λ'_d, λ'_l) while keeping $\lambda' = \lambda$.

4.3.4 p-restricted multicast with MultiSphinx

As specified so far, Rollercoaster uses the unmodified Loopix protocol. However, even though Rollercoaster spreads the work of sending a multicast message more evenly across the network than sequential unicast, payload messages and ACKs are still demanding for nodes' send queues. In this section, we consider a modification to the Loopix protocol that further improves multicast performance: namely, we allow some mix nodes to *multiply* one input message into multiple output messages, which may be sent to different recipients. The naïve mix-multicast we considered in Section 4.2.2 allows arbitrary multiplication factors. Here we show how to make mix-node-supported multicast safe by restricting the multiplication factor to a fixed constant p. We call this approach *p-restricted multicast* where clients can send p messages inside one MultiSphinx package; with $p = 1$ this scheme is identical to the regular Rollercoaster.

In p-restricted multicast, only mix nodes in one designated layer may multiply messages. In our design, we perform multiplication in the middle layer (layer 2 of 3) and we refer to these mix nodes as *multiplication nodes*. To ensure unlinkability of mix nodes' inputs and outputs, *every* message processed by a multiplication node must result in p output messages, regardless of the message type or destination. Mix nodes in other layers retain the standard one-in-one-out behaviour of Loopix. Since layer 3 of the mix network needs to process p times as many messages as the earlier layers, layer 3 should contain p times as many mix nodes as layers 2.

We use the parameter p for p-restricted multicast and k for the schedule algorithm. These can be chosen independently of each other. However, for simplicity and practical interdependence we often set both to the same value $k = p$.

Effectively, p-restricted multicast allows p messages to different recipients to be packaged as a single message up to p times the size. Sending fewer but larger messages allows for lower power consumption on mobile devices, as discussed in our application requirements in Section 4.1 and Chapter 3. We show in our evaluation in Section 4.4.5 that p-restricted multicast allows choosing much larger λ values while maintaining low latency allowing mobile devices to stay longer in energy-efficient states. Further, since collaborative editing update messages (e.g. indicating a single word has been inserted) can be small, bundling them together can reduce the overall overhead.

The MultiSphinx message format

Loopix encodes all messages using the *Sphinx* message format [44], which consists of a header M containing all metadata and an encrypted payload δ. Using the header, each mix node n_i derives a secret shared key s_i. Due to the layered encryption of the header and payload, an adversary cannot correlate incoming and outgoing packets when observing mix nodes. Our construction is based on the improved Sphinx packet format [20] which uses authenticated encryption (AE) instead of the wide-block cipher *LIONESS* [4] of the original design. In particular, we use a stream cipher C in an encrypt-then-MAC regime and require that without the knowledge of the key, the generated ciphertext is indistinguishable from random noise (which is believed to be the case for modern ciphers such as AES-CTR). Every hop verifies integrity of the entire message to prevent active tagging attacks. The improved Sphinx packet format satisfies the ideal functionality of Sphinx [83]. The per-hop integrity checks of the entire message come at the cost of lacking support for anonymous reply messages, but these are not used by Loopix.

Sphinx assumes that each input message to a mix node results in exactly one output message. In order to support p-restricted multicast we introduce the MultiSphinx message format, which can wrap p messages. A MultiSphinx message is unwrapped at a designated mix node, and split into p independent messages. For anyone other than the designated multiplication node, MultiSphinx messages are indistinguishable from regular Sphinx packets. We now describe the MultiSphinx design for $p = 2$ by describing the creation and processing of these messages. The detailed construction and processing is formalised in Appendix A.5.

For $p = 2$, the sender waits until its message queues (payload, drop, loop) have released two messages. The sender then combines their payloads δ_A, δ_B and recipients U_A, U_B into a single message that is inserted into the mix network, as shown in Figure 4.3. As we want to fit both payloads and two headers into our message to the multiplication node, $|\delta_A|$ and $|\delta_B|$ must be smaller than the global Sphinx payload size.

The combined message is sent via a mix node n_0 in the first layer to the designated multiplication node n_1, where its inner messages are extracted and added to its input buffer. The inner message containing δ_A will be processed by n_1 and routed via $n_{2,A}$ to the recipient n_A (and similarly for B). The multiplication node derives the secret key s_1 from the incoming message's header and additional secret keys $s_{1,A}, s_{1,B}$ from the headers of the inner messages. We omit provider nodes.

The sender first computes all secret keys. Using these secret keys it encrypts the payloads δ_A, δ_B between the recipients and the multiplication node. However, the resulting encrypted payloads are smaller than the regular Sphinx payload lengths.

To ensure all messages have the same size, we use a pseudo-random function (PRF, e.g. HMAC) ρ to add padding to the encrypted payloads $\delta_{1,A}$ and $\delta_{1,B}$. ρ is keyed with the shared secret s_1 and the payload index (`A` or `B`) so that the padding is unique. The resulting payloads have the format $\delta'_{1,A} = \delta_{1,A} \parallel \rho(s_1 \parallel \texttt{A})$ (and similarly for B). Now the sender computes the headers and MACs along the path from the multiplication node to the recipients by simulating the decryption of the payload at each step. This results in two Sphinx headers $M_{1,A}$ and $M_{1,B}$. Finally, we create the message for the path from the sender to the multiplication node using the regular Sphinx construction. We set the payload of that message to the concatenation $\delta_{combined} = M_{1,A} \parallel \delta_{1,A} \parallel M_{1,B} \parallel \delta_{1,B}$. Appendix A.5 contains pseudocode for this construction.

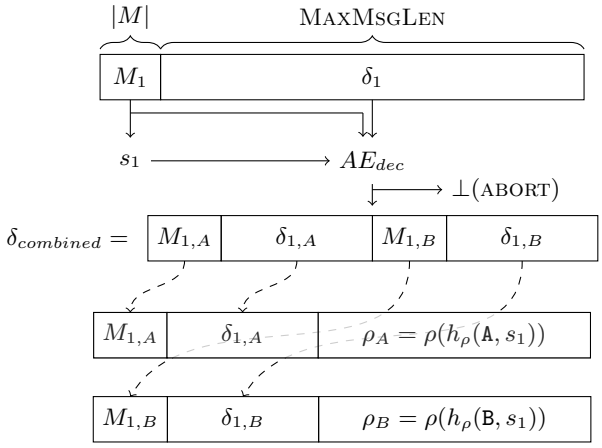

Figure 4.4: Processing of a MultiSphinx message at the multiplication node n_1 resulting in two outgoing messages that are sent then re-queued for processing.

The processing of incoming messages at the multiplication node differs from other nodes and is illustrated in Figure 4.4. First, the payload is decrypted and split into the message headers and payloads. Then, the payloads are deterministically padded using the PRF ρ as described above. To ensure that the messages are hard to correlate, they are added to the node's input buffer, decrypted again (now deriving secrets $s_{1,\{A,B\}}$), and delayed independently as defined by their individual delay parameter.

Anonymity of MultiSphinx

All MultiSphinx messages (before and after the multiplication node) have the same header length and payload length as regular Sphinx messages. Sphinx headers do not leak the number of remaining hops and the ciphertext is indistinguishable from random noise. Therefore, MultiSphinx messages are indistinguishable from regular Loopix messages (Lemma 31, Appendix A.6). At the same time, the multiplication node maintains the unlinkability between the incoming messages and outgoing messages as these are delayed independently. An adversary might also corrupt mix nodes. Even in this case they do not gain advantage over regular Sphinx message with regards to sender and recipient anonymity and unlinkability (Theorem 35, Appendix A.6). These results also hold for active adversaries with the capabilities from the original Loopix paper(Theorem 41, Appendix A.6).

If an adversary controls a p-restricted multiplication node and c_3 of the n_3 mix nodes of the third layer, they can trace some messages from their multiplication to their delivery at providers. On the basis that the p recipients of a MultiSphinx message are likely to be members of the same group, the adversary then has a chance to guess that any two of the users from these providers share a group membership. In Theorem 39 (Appendix A.6) we show that the probability of correctly guessing two group members given a group message is less than $(1 - (\frac{n_3 - c_3}{n_3})^{p-1}) \cdot \frac{|\mathcal{P}|^2}{|\mathcal{U}|^2}$ if all $|\mathcal{U}|$ users are evenly distributed among $|\mathcal{P}|$ providers. This attack is prevented if the multiplication node *or* all but one of the chosen nodes in the third layer are trustworthy—in contrast, standard Loopix requires only that any mix node on the message path is trustworthy. MultiSphinx does not leak any information regarding group sizes (Theorem 37). Appendix A.6 contains theorems and proofs for our claims. In

addition to these properties, it is possible to achieve sender anonymity by first forwarding the message to a trusted group member. The sender can prove its membership through a shared group secret.

4.3.5 Further optimisations

The schedule computed by GenSchedule in Algorithm 1 delivers the first messages to the nodes at the beginning of the provided recipient list U_{recv}. These nodes will always act as the forwarding nodes for a given sender s. To better balance these among all group members, one can shuffle the list based on a *nonce* value that is part of the message. This variant is described in Algorithm 2. As the GenScheduleRand algorithm is still deterministic and the nonce is part of the Rollercoaster header, each node reconstructs the same schedule.

Algorithm 2 Creating a pseudo-randomised schedule for a given nonce

1: **procedure** GenScheduleRand(s, U_{recv}, k, *nonce*)
2: $R \leftarrow$ NewPrng(*nonce*)
3: $U'_{\text{recv}} \leftarrow$ R.shuffle(U_{recv})
4: **return** GenSchedule(s, U'_{recv}, k)

Further optimisation is possible if different sub-groups display different levels of activity and connectivity. For example, if there is a small, active sub-group communicating while the rest of the group remains passive, it is more important for messages to travel faster between active nodes to support swift, effective collaboration. Active nodes can often be assumed to be more likely to be online. Agreeing on the full order is no longer possible through a single nonce value. However, the source can compute schedules for a set of randomly chosen nonces, evaluate the generated schedules against its information about the group members, and choose the nonce that results in the schedule with the most desirable properties.

4.4 Evaluation

For the empirical evaluation we developed a mix network simulation tool that provides fully reproducible results. First, we discuss the behaviour and results of the Rollercoaster scheme in an ideal scenario where all participants are online throughout. Second, we discuss the impact of offline nodes and how this is addressed by the fault-tolerant variant of Rollercoaster. Finally, we discuss the impact of multi-group membership, sending multiple messages at once, and p-restricted multicast.

4.4.1 Methodology

Since the real-world performance of Loopix has been practically demonstrated [110] we run a simulation instead of an experiment on a real network. This provides clear practical advantages: First, it allows us to eliminate external influences such as network congestion due to unrelated traffic or CPU usage by other processes. Second, the simulated time can run faster than real-time, allowing us to gather significantly more results using less computational resources. Third, it makes monitoring and categorising traffic easier as packets and node state can be inspected. Finally, by initialising the PRNG with a fixed seed, the results in this section are fully reproducible.

The simulator runs the entire mix network on a single machine, with nodes communicating through shared memory simulating a network. It instantiates objects for each participating user, provider, and mix node. All objects implement a `tick()` method in which they process incoming messages and mimic the designed node behaviour such as delaying and forwarding packets. As we are primarily interested in the traffic behaviour, no actual encryption is performed. The original Loopix paper showed that the queueing time and per-hop delays dominate the message delay, and that CPU time for cryptographic operations is insignificant in comparison. Similarly, the network delay is negligible.

For the final evaluation we ran 276 independent simulations, covering more than 992 160 hours of simulated node time in less than 209 hours of CPU core time. This is a relative speed up by factor 4 500× compared to a real network experiment of the same scope. The scenarios were first specified using Python notebooks and saved as *pickle* object files. The simulations were then executed using the *Peta 4* high-performance computing cluster provided by the *Cambridge Service for Data Driven Discovery* (CSD3). In every simulation step the application (see below) measures the message latency d_{msg} of each delivered message between the original source and each recipient. The resulting data was persisted and extracted for analysis. We verified that our simulator behaves faithfully to the Loopix implementation by reproducing a latency distribution graph from the original paper [110, Figure 11], as shown in Appendix A.7. Our simulator is implemented in less than 2 000 lines of Python code including tests and is available as an open-source project: `https://doi.org/10.5281/zenodo.10606414`. The simulator and analysis scripts passed the USENIX Security artefact evaluation process.

The network simulator assigns 16 users to each provider. We set the Loopix rates $\lambda_p = \lambda_d = \lambda_l = 2/\text{s}$ for the client nodes and the delay rate $\lambda_\mu = 3/\text{s}$. Hence, the overall sending rate of the clients is $\lambda = 6/\text{s}$. This meets the requirement $\lambda/\lambda_\mu \geq 2$ that is suggested by the Loopix paper [110, p. 1209, $\lambda_\mu = \mu$]. The mix network consists of 3 layers containing 3 mix nodes each (mix loop injection rate $\lambda_M = 2/\text{s}$). All simulations are run with a granularity of 10 ms per tick. The simulated time span for all configurations is 24 h.

The application behaviour is modelled by a Poisson process. On average every 30 s one of the online nodes sends a single message to all other group members. We account for participation inequality [98] by dividing the group using an 80/20 rule: 20% of users in the group send 80% of all messages, and vice versa.

4.4.2 Always-online baseline

For a group of size 128, the average latency is reduced from 34.9 s in sequential unicast to 7.0 s (8.3 s for group size 256) in Rollercoaster with $k = p = 2$. This fulfils our application requirements that were derived from the user study concerning delay in collaborative applications [71]. The results are compatible with our analytical results as discussed in Section 4.3.3. For Rollercoaster not only is the average latency low, but most of the latency distribution falls within fairly tight bounds – that is, very large latencies are rare. Figure 4.5 shows the latency achieved by the Rollercoaster scheme with and without p-restricted multicast for different percentiles and compares them to unicast. For a group with 128 members the 99th percentile p_{99} for Rollercoaster is 12.3 s (p_{90}: 9.9 s) whereas in unicast it is 75.6 s (p_{90}: 60.8 s). The wide "body" of the distribution for sequential unicast is a result of the fixed send rate of the sender. Hence, the latency is dominated by the time spent in the source's payload send queue. We provide detailed histograms in Appendix A.3.

4.4.3 Fault tolerance

The evaluation of the fault tolerance properties requires a realistic model of connectivity of mobile devices. For this we processed data from the *Device Analyzer* project [144] that contains usage data of volunteers who installed the Android app. The online/offline state of a device is derived from its trace information regarding network state, signal strength, and airplane mode. We limit the dataset ($n = 27\,790$) to traces that contain connectivity information ($n = 25\,618$), cover at least 48 hours ($n = 20\,117$), and have no interval larger than 12 hours without any data ($n = 2\,772$).

Inspecting the traces we identify three archetypes of online behaviour. The first group is online most of the time and is only interrupted by shorter offline segments of less than 60 minutes. Members of the second group have at least one large online segment of > 8 hours and are on average online 50% or more of the time. Finally, the third group is online less than 50% of the time with many frequent changes between online and offline states. As the dataset is more than five years old we decided to use the characteristics of these groups to build a model. Using a model allows us to extrapolate offline behaviour into scenarios with increased connectivity. In the model following the parameters of the original dataset, the fraction of all users' time spent online is 65%. In a second and third model with increased connectivity, we increase this percentage to 80% and 88%, respectively, while preserving the behaviour of the archetype groups. The generated models are visualised in Appendix A.8.

For our discussion of offline behaviour we refine our previous definition of message latency d_{msg}: we ignore all latencies where the intended recipient was offline when the message was placed into their inbox by the provider node. This metric captures the subset of latencies where the intended recipient was online when the message was placed into their inbox by the provider node. We exclude all results where the metric evaluates to $\perp$.

$$d_{\text{online}} = \begin{cases} T_{\text{recv},B} - T_{\text{send},A} & \text{if } B \text{ is online at } T_{\text{recv},B} \\ \perp & \text{otherwise} \end{cases} \tag{4.5}$$

This change has the practical benefit of excluding outliers. More importantly, fast delivery to an offline user has no real-world benefit. Instead, a good multicast algorithm should optimise the delivery to all nodes that are active and can actually process an incoming message. The source might go offline at any time regardless of outstanding messages.

Without fault tolerance, the presence of offline nodes greatly increases the 99th percentile (p_{99}) for Rollercoaster (RC) to more than 10 000 s for a group of 128 members. The fault-tolerant variant (RC-FT) reduces the 99th percentile to less than 21.9 s (p_{90}: 18.0 s). In unicast p_{99} latency is 103.3 s (p_{90}: 61.9 s). Figure 4.6 shows that the fault-tolerant variant generally outperforms unicast at various percentiles. We provide detailed histograms in Appendix A.3.

4.4.4 Multiple groups and message bursts

Users might be part of multiple groups, which increases their burden of distributing messages. In this evaluation we assign 128 users to a growing number of groups. Figure 4.7 shows that the number of group memberships has little impact on performance of Rollercoaster both for online and offline scenarios.

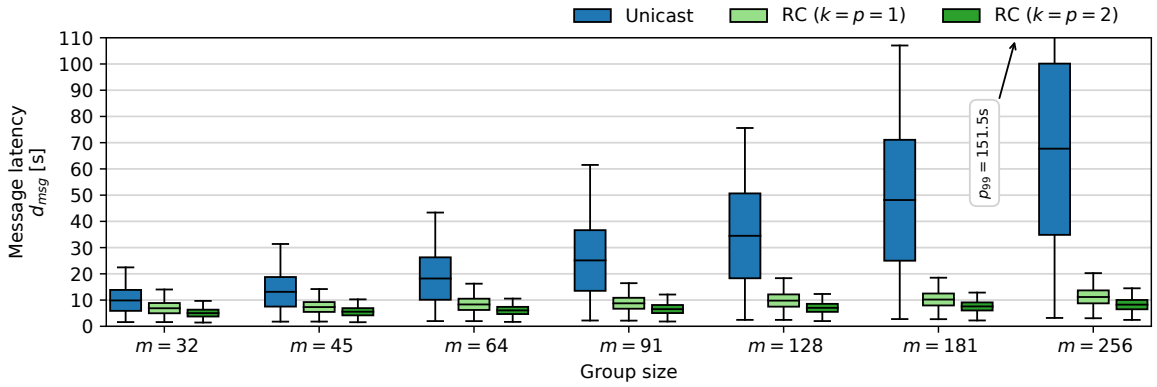

Figure 4.5: This box plot shows the distributions of message latency d_{msg} for increasing group sizes for the strategies *naïve sequential unicast* and *RollerCoaster (RC)*. The RollerCoaster strategies show different k and p parameters. The boxes span from the first quartile to the third quartile (middle line is the median) and the whiskers indicate the 1st and 99th percentile.

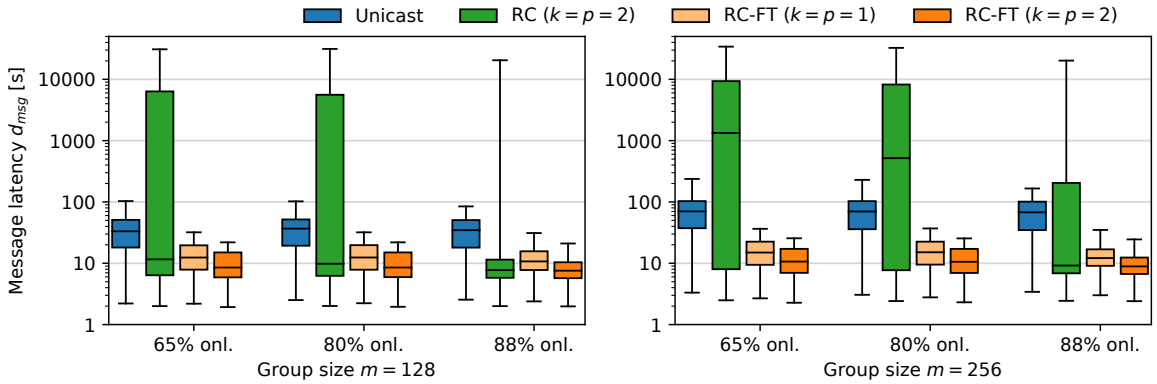

Figure 4.6: The distribution of message latency d_{msg} for different offline scenarios. From left to right the strategies are Unicast, RollerCoaster without fault-tolerance (RC), and RollerCoaster with fault-tolerance (RC-FT). Boxes and whiskers as in Figure 4.5.

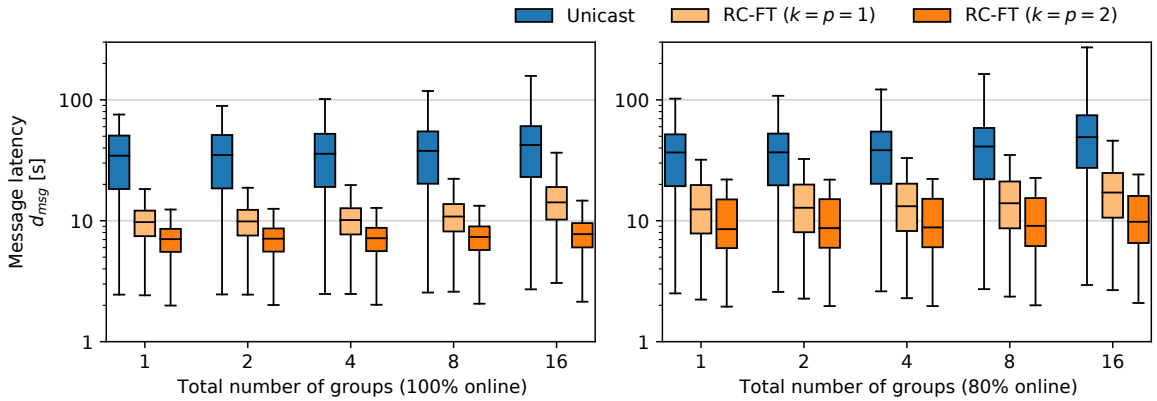

Figure 4.7: Message latency d_{msg} for an increasing number of groups for 128 users (every user is member of every group). Boxes and whiskers as in Figure 4.5.

Similarly, users might be sharing large payloads (e.g. images) or sending multiple updates at once. Both translate into many messages being scheduled for distribution at the same time, which risks overwhelming the payload queue. Figure 4.8 shows that Rollercoaster can handle many more messages sent in bursts than unicast. We observed that with unicast and some Rollercoaster configurations some nodes had indefinitely growing send buffers as the simulation progressed. The effect of this can be seen by the higher message latencies for $b = 32$. This threshold is higher for p-restricted multicast.

4.4.5 p-restricted multicast

In this evaluation we show that p-restricted multicast allows us to drastically lower the sending rates $\lambda_{\{p,d,l\}}$ of the clients while achieving similar performance. A low sending rate is desirable as it allows the radio network module to return to standby and thereby saving significant battery energy on mobile devices. Figure 4.9 shows that just increasing k (left) has negligible or even negative impact, while increasing k and p together (right) allows for lower sending rate λ while maintaining good enough performance. We decrease λ_μ accordingly to maintain the $\lambda/\lambda_\mu \geq 2$ balance (Section 4.4.1) which increases the per-hop delays.

4.5 Summary

In this chapter we presented Rollercoaster, a protocol to efficiently send messages within groups on top of mix networks such as Loopix. It works by involving all group members in the distribution process and thus improves the overall balance between real messages and cover messages. This reduces sending rates which in turn provides energy benefits on mobile devices as we saw in the previous chapter. Importantly, Rollercoaster (excluding the optional MultiSphinx extension) works without any modification of the underlying anonymity network allowing applications that use it to share the anonymity set of the entire network.

Building on the basic idea of Rollercoaster, we added fault-tolerance to gracefully handle clients that are temporarily offline, as this is common with smartphones. We also added an optional MultiSphinx extension that allows even faster message distribution by duplicating messages as they travel through the network. In our evaluation we showed that Rollercoaster outperforms unicast for both average latency and outliers—especially so as the group sizes grow. Our fault-tolerance mechanism ensures that Rollercoaster maintains its advantages even if some group members are offline. Finally, we show that it also works well when traffic appears in bursts or when users are member of multiple groups.

Rollercoaster can be an important building block for applications such as group messaging and collaborative document editing over an anonymity network. However, just a multicast primitive by itself is not sufficient to build secure and decentralised applications. In our conclusion chapter we discuss how Rollercoaster can be integrated with other research and highlight important open questions for future work.

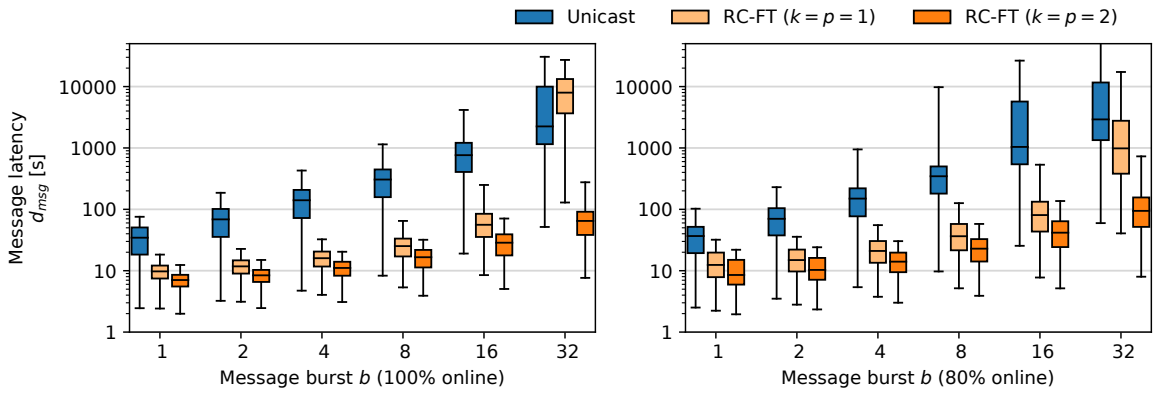

Figure 4.8: Message latency d_{msg} for applications that send b messages at once. The group size is $m = 128$. Boxes and whiskers as in Figure 4.5.

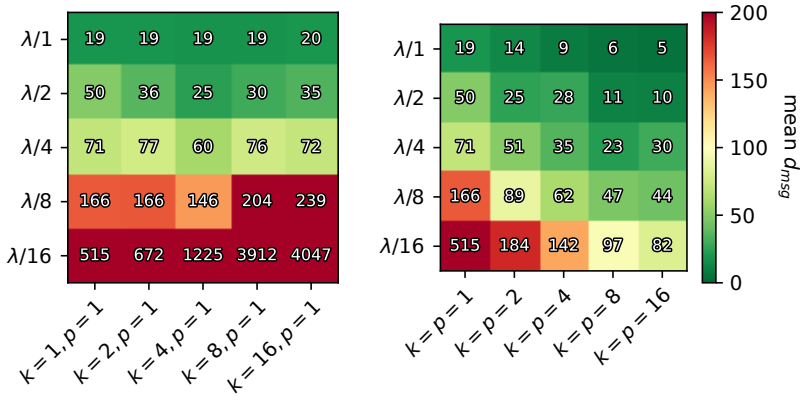

Figure 4.9: Heatmaps showing the mean message latency for reduced sending rates (y-axis) and different ROLLERCOASTER parameters (x-axis). In the left graph only the logical branch factor k is increased. In the right graph the multicast factor p is increased at the same time. Group size is $m = 128$ and 80% online. We present more configurations in Appendix A.2.

Chapter 5

Key stretching and deniable encryption using the Secure Element on smartphones

In the previous chapters we discussed protocols for securing communication between devices and making sure that they do not leak sensible metadata. However, this leaves open the question of how we can secure data at rest on the devices so that the stored content remains confidential in case a device is lost or stolen. This local data might be the synchronised document versions of a collaborative editing app or the message history of a chat app. At first glance this problem appears already solved, as most smartphones offer encryption of the entire phone storage using either biometrics or passphrases which allow the legitimate user to unlock the device and the stored data. However, "just encrypting" the data might not be sufficient for situations where users might be compelled to unlock their device. These could happen to a person who is suspected of contacting a journalist or an NGO employee at a border crossing. In such situations, the user should be able to plausibly argue that there is no interesting information stored on the device at all. This property is called plausible deniability. We use plausibly-deniable storage for the CoverDrop app (Chapter 6) where users need the ability to deny that they previously used the whistleblowing feature that is integrated into a news app.

Implementations for plausibly-deniable storage typically rely on memorable passphrases, as the alternatives are unsuitable: biometrics can be collected from users involuntarily and long, complex passphrases are not user-friendly. Therefore, we need to use key stretching [77] techniques to derive a strong, high-entropy cryptographic key from a relatively low-entropy passphrase. Modern password hashing functions such as PBKDF2 [93], scrypt [105], and Argon2 [24] are typical choices today that use computation and memory-hard functions to make it costly to try all possible passphrases. However, an attacker can still increase their brute-force throughput by using more computers. Memory-hard functions, as used by Argon2, can mitigate some of these risks, such as application-specific integrated circuits (ASICs), but still leave short passphrases vulnerable to attacks.

In this chapter we design and evaluate a practical key stretching scheme for mobile devices that uses the Secure Element (SE) of a smartphone to effectively rate-limit the guess rate of an attacker.

We present the two variants LongSloth and RainbowSloth, for Android and iOS respectively, that can be integrated by developers today on these platforms without any modifications to the hardware or operating system. In particular, an attacker does not gain any meaningful advantage by using more computers, as the verification of passphrases is dependent on a cryptographic operation with a non-extractable key inside the SE. Based on these we construct HiddenSloth, a plausibly-deniable encryption scheme for mobile applications on both platforms. On Android we extend it with a ratchet operation so that it remains plausibly-deniable even when an adversary captures multiple snapshots of the encrypted storage.

Plausibly-deniable storage for desktop computers was first implemented in 2000 with StegFS [89] and is supported by the popular VeraCrypt [113] (virtual) disk encryption software. Proposed solutions for mobile devices, such as MobiFlage [124], MobiCeal [31] and the work by Liao et al. [87], require changes to the operating systems which makes them unlikely candidates for widespread adoption. Also, these solutions do not provide strict time guarantees like Sloth. For online services, the problem of parallel offline brute-force attacks is typically addressed through rate-limiting logic in the backend code. This includes standard login forms over HTTPS as well as password-authenticated key exchange (PAKE) protocols such as OPAQUE [74]. However, communication with external services leaves network traces which can threaten plausible deniability, as the adversary can use them as proof of active usage.

This chapter is based on our paper "Sloth: Key Stretching and Deniable Encryption using Secure Elements on Smartphones" [70] which is currently under review. For this chapter I adapted the text, figures, and plots from the Sloth paper to fit with the dissertation. I led the design of the protocol, implemented the prototype, conducted all experiments, performed the data analysis, and was the primary author of the text. My co-author Alberto was the main author of the formal proofs which are included in the appendices. Alberto, Sam, and Alastair contributed towards the development of the ideas and their presentation.

5.1 Secure Elements on Android and iOS

Before we introduce the Sloth schemes in Sections 5.2f, we discuss typical hardware support for secure cryptographic operations on smartphones and highlight the limitations of the APIs on both Android and iOS. We then survey the prevalence of SEs on modern smartphones which shows that they are widely available with growing support in newer models.

5.1.1 Background

Hardware support for executing code in a secure context is available in many modern smartphones and Trusted Execution Environments (TEEs) were the first architecture that achieved wide integration. TEE implementations, such as ARM TrustZone [109], run on the main chip, but in a privileged context so that even a compromised kernel cannot manipulate them. However, TEEs generally come with limited protection against side-channel and physical attacks. Secure Elements (SE) are standalone components with their own dedicated CPU, memory, and storage. This provides stronger isolation and protection against physical attacks.

Android allows developers to create and use keys via its API, which abstracts a *Keymaster* interface to manage keys. From API level 23 (Android 6.0 M, released 2015) application developers can verify that keys are stored inside secure hardware. For phones released around 2015, this would typically mean that they are managed by a Keymaster implementation inside a TEE. However, from API level 28 (Android 9 P, released 2018), Android supports the StrongBox Keymaster implementation which must be implemented using an SE [56, 58]. Google supports StrongBox in its flagship models since the Pixel 3 release in 2018. Other devices might have included SEs before this, but app developers would not have been able to use them.

Apple refers to SEs in their devices as a *Secure Enclave* and they are supported in iOS devices since the iPhone 5S (released 2013) [8]. From the beginning, they have been used to protect biometric data and secure device encryption. However, they only were exposed to app developers with iOS 13 (released 2019) [12].

While TEEs and SEs allow for a much smaller implementation and attack surface, they are not impenetrable. Researchers successfully extracted private keys from TrustZone implementations from Qualcomm [116] and Samsung [120]. Intel's SGX extension is similar in broad terms to TrustZone where there are many documented attacks [99].

5.1.2 APIs and limitations

Code that runs on the SE is (by design) executed without any control by the operating system. Therefore, platforms do not allow developers to run custom code. Instead, they offer access to specific cryptographic operations through APIs. These APIs serialise the user request and send it to the SE where it is parsed and executed. The result is returned similarly.

We discuss the API on iOS first as it consists of very few operations. This provides a minimal attack surface, but as we will see later, it also stands in the way of efficient software implementation. At the time of writing, iOS only supports storing private P-256 Elliptic Curve (EC) keys [13]. Once created inside the SE, the API provides methods to sign data, output the corresponding public key, and perform key agreement via Elliptic Curve Diffie-Hellman (ECDH). The API also provides methods for encrypting and decrypting data. However, the documentation is not clear on whether this happens entirely within the SE, or whether the ECDH result is shared with an AES engine outside the chip.

The Android API offers more operations and key types to developers. It uses the Keymaster API which abstracts the actual key handling from the user. Device vendors implement the Keymaster HAL which serialises the API calls and communicates with the backend. This backend could be a service running in a TEE or SE. We are interested in the StrongBox Keymaster implementation which requires using a SE [58]. It guarantees the availability of the following algorithms: RSA-2048, AES-128/256, ECDSA P-256, ECDH P-256, HMAC-SHA256, and 3DES. The availability of symmetric cryptography allows us to come up with a simpler design for Android devices. However, it also increases the complexity of the implementation that device manufacturers have to provide. Android recently added a new API `setMaxUsageCount` for OS version 12+ (Android S, API 31) that deletes the key after a given number of operations. While it would be convenient, it is implemented at the OS level[1], as

[1] `https://cs.android.com/android/platform/superproject/+/master:system/keymint/common/src/tag.rs;l=314-333;drc=ed657df7c7b329fb3d26eb0ce88af92594245ae8`

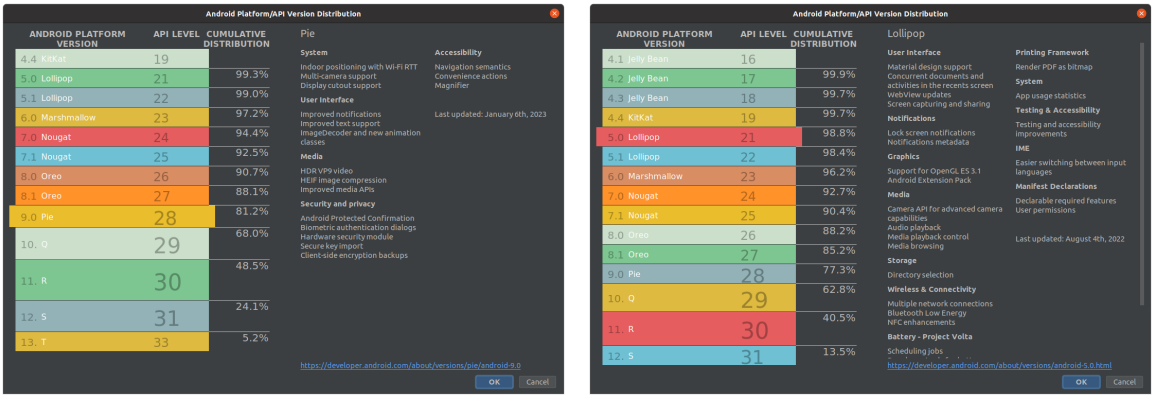

Figure 5.1: Distribution of Android versions as shown in the Android Studio IDE for January 6th, 2023 (left) and August 4th, 2022 (right).

StrongBox does not have sufficient persistent per key storage and thus cannot manage respective counters internally.

5.1.3 Support for Secure Elements on iOS devices

The availability of SE functionality for end-user apps is determined by hardware support, i.e. whether the device has an SE, as well as platform support for the respective API. As discussed, Apple first added secure enclaves to their iPhone 5S in 2013 and an API was added in iOS 13 which was released in 2019. Since the iPhone 5S only supported iOS up to version 12, all devices with iOS 13 (platform support) also contain a Secure Enclave (hardware support). This is supported by the fact that there is no API to check for the presence of a Secure Enclave. The App Store statistics for May 2022 show that 82% of all iPhones use iOS 15 and 14% use iOS 14 [9]. No data is given for iOS 13 or before. Therefore, at least 96% of all iPhones devices expose SE functionality to developers and are compatible with our Sloth scheme.

5.1.4 Support for Secure Elements on Android devices

The situation for the Android ecosystem is more complex as devices are manufactured by different vendors with different hardware as well as changes to the operating system. This can lead to situations where Android devices have an SE, but do not offer API access, or where a device's API is compatible, but has no SE. We use the overall distribution of active Android versions as an upper boundary for platform support. We then execute test code on dozens of real devices to determine hardware support.

For the platform support we use the API Version Distribution that is shown in the "New Project" wizard in the Android Studio IDE. Figure 5.1 shows the API distribution that is shown when creating new project in the Android Studio IDE when it was last updated on January 6th, 2023 and for the previous update in August 4th, 2022. In particular, Figure 5.1 shows that 97.2% of devices run API level 23 (Android M) or higher and hence provide the API to check whether a key is backed by a TEE or SE. Furthermore, 81.2% of devices run API level 28 (Android P) or higher and hence provide the StrongBox API that can enforce storage in an SE.

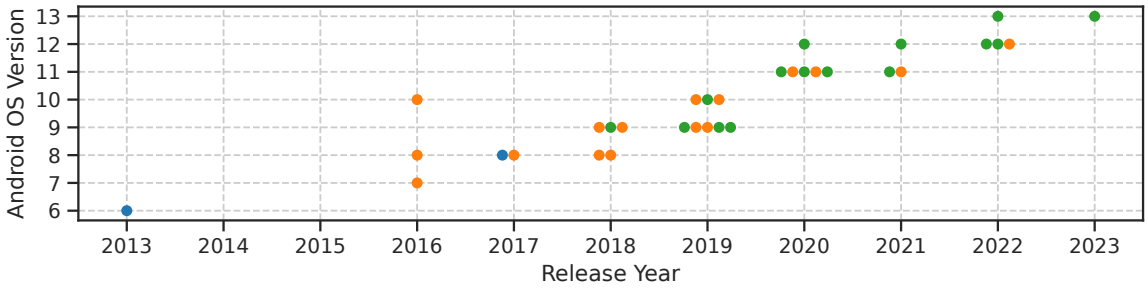

Figure 5.2: Swarm plot of all surveyed Android devices. We use green ● for StrongBox support, orange ● for TEE support, and blue ● for no support for hardware-backed keys at all.

For hardware support, we use the AWS Device Farm [3] which allows remote access to real devices in a data centre. We compile a test application and upload it for execution on all selected device types. Our test code performs multiple checks. First, we create different key types and then check via `KeyInfo#isInsideSecureHardware` if it is stored in secure hardware. A positive answer indicates that the device has a TEE *or* SE. Then, we read the `PackageManager` feature flag for StrongBox support: `FEATURE_STRONGBOX_KEYSTORE`. Finally, we create different key types with the `setIsStrongBoxBacked` property that enforces storing them in an SE and checking for failures. Figure 5.2 and Table 5.1 summarise our results.

These results show that the vast majority of devices released after 2020 provide access to an SE. And all of those have at least TEE support. The earliest device with SE support is the Google Pixel 3 which was released 2018. The release year and Android OS version strongly correlate meaning that newer devices indeed run more recent OS versions. In our sample, all devices with OS version 12+ (Android S, API 31) offer an SE. However, based on the available API distribution (Figure 5.1), only 14% of active handhelds run this OS version. For an estimate of the overall availability, we take the relative presence of SEs for each OS version and weight it based on the API distribution. This yields an estimate of 45% devices supporting SE globally for data from January 2023 (up from 39% in August 2022). However, this value will differ between countries and we believe it will continue to improve as more device features, such as contactless payments, rely on SEs.

Note that the percentages are cumulative and in order to get the actual percentage for a specific version number, e.g. API 9 in January 2023, we calculate $81.2\% - 68.0\% = 13.2\%$. For calculating the estimated share of supported devices we first compute the prevalence of SE support per API level using Table 5.1. This yields the following data: API 9 (50.0%) API 10 (25.0%) API 11 (57.1%) API 12 (80.0%).

We then calculate for January 2023 as follows: $13.2\% \times 50.0\% + 19.5\% \times 25.0\% + 24.4\% \times 57.1\% + 24.1\% \times 80.0\% \approx \mathbf{44.7\%}$. And for August 2022 likewise: $14.5\% \times 50.0\% + 22.3\% \times 25.0\% + 27.0\% \times 57.1\% + 13.5\% \times 80.0\% \approx \mathbf{39.1\%}$.

Device Model	OS Version	Release Year	TEE	SE
ASUS Nexus 7 - 2nd Gen (WiFi)	6	2013		
Google Pixel	7	2016	✓	
Google Pixel 2	8	2017	✓	
Google Pixel 3	9	2018	✓	✓
Google Pixel 4 (Unlocked)	10	2019	✓	✓
Google Pixel 4a	11	2020	✓	✓
Google Pixel 5 (Unlocked)	12	2020	✓	✓
Google Pixel 6 (Unlocked)	12	2021	✓	✓
Google Pixel 7	13	2022	✓	✓
LG Stylo 5	9	2019	✓	
OnePlus 8T	11	2020	✓	
Samsung Galaxy A7	8	2016	✓	
Samsung Galaxy A10s	10	2019	✓	
Samsung Galaxy A13 5G	11	2021	✓	
Samsung Galaxy A40	9	2019	✓	
Samsung Galaxy A51	10	2019	✓	
Samsung Galaxy A71	11	2020	✓	
Samsung Galaxy J7 (2018)	8	2018	✓	
Samsung Galaxy Note 10	9	2019	✓	✓
Samsung Galaxy Note 20	11	2020	✓	✓
Samsung Galaxy S8 (T-Mobile)	8	2017		
Samsung Galaxy S9 (Unlocked)	9	2018	✓	
Samsung Galaxy S10	9	2019	✓	✓
Samsung Galaxy S21 Ultra	11	2021	✓	✓
Samsung Galaxy S22 5G	12	2022	✓	✓
Samsung Galaxy S23	13	2023	✓	✓
Samsung Galaxy Tab A 10.1	10	2016	✓	
Samsung Galaxy Tab S4	8	2018	✓	
Samsung Galaxy Tab S6 (WiFi)	9	2019	✓	✓
Samsung Galaxy Tab S7	11	2020	✓	✓
Samsung Galaxy Tab S8	12	2022	✓	✓
Sony Xperia XZ3	9	2018	✓	
Xiaomi 12 Pro	12	2022	✓	

Table 5.1: All 33 surveyed devices from the AWS device farm. The *Device Model* is the name provided by the AWS API. The *OS Version* refers to the tested OS version and the device might be available with different versions.

	Algorithms and schemes
KDF	Key derivation function
H	Hash function
Ξ	Key stretching scheme
Δ	Deniable encryption scheme
	Parameters
λ	Security parameter for computational security
l	Length of the ω_{pre} pre-secret in LongSloth
n	Count of ω_{pre} pre-secrets in RainbowSloth
	Variables
pw	The user password
m	The min-entropy of the password distribution
$\pi \in \Pi$	Storage state (space)
$\psi \in \Psi$	State (space) inside the SE
h	Key handle for the SE
ω	An intermediate secret
k	The final derived secret key

Table 5.2: We use these symbols for algorithms, parameters, and variables in our descriptions throughout this chapter.

5.2 The Sloth key stretching schemes

We introduce our Sloth schemes by first giving a high-level overview of the system (Section 5.2.1) and our threat model (Section 5.2.2). Then we present the key stretching schemes LongSloth (Section 5.2.3) for Android and RainbowSloth (Section 5.2.4) for iOS.

Notations. Throughout the chapter, let λ be a freely chosen but fixed security parameter. We write $x.y \leftarrow z$ to denote the assignment of value z to the field y of a named-tuple x. Table 5.2 summarises the symbols that we use in this chapter.

5.2.1 System overview

Figure 5.3 provides a high-level overview common to all our Sloth schemes. Sloth distinguishes between two execution spaces: the general device space and the SE. We assume operations performed on the main device may be controlled by the adversary, e.g. using a local kernel exploit, while we assume the SE is a separate piece of hardware and remains secure (Section 5.2.2). The user inputs a password, pw, in the user space. The sloth schemes start by pre-processing pw using the state $\pi \in \Pi$ where π is accessible in user space; the output of this pre-processing step, ω_{pre}, is given as input to the SE. The SE maintains its own secure state, $\psi \in \Psi$, with cryptographic keys inaccessible to the device. Each key maintained in ψ is associated with a unique (and public) key handle h. The SE accesses ψ and executes a cryptographic operation SE-OP; its output ω_{post} is returned to the user space for

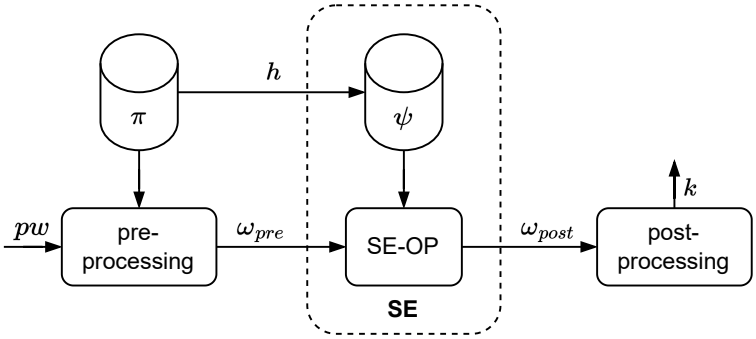

Figure 5.3: Overview of the abstract design of our SLOTH schemes. The user password, pw, is first pre-processed on the device with access to device state π, producing ω_{pre}; the SE executes a cryptographic function over ω_{pre} using its internal (hidden) state ψ; the SE's output, ω_{post}, is then returned to the device; the device can then post-process ω_{post}, for example by applying a key derivation function, resulting in k as the final output.

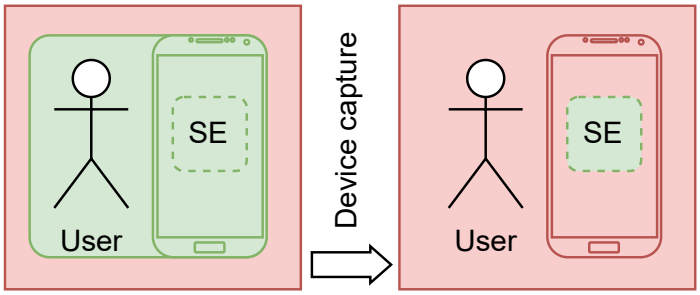

Figure 5.4: The left side illustrates the adversary reach before device capture where the user can safely operate the smartphone. The right side illustrates the state after capture where only the Secure Element resists the adversary who gained full-access to the rest of the phone.

further post-processing to derive the final key k. The key k can then be used for authentication, full disk encryption, or deniable encryption protocols.

5.2.2 Threat model

We assume an adversary can physically capture the device. For instance, they find a lost device on the street or stop the user at a border crossing. Before the capture, we assume that the device can be securely and confidentially operated by the user. After the capture, only the SE resists full-access by the adversary. We illustrate this in Figure 5.4.

The adversary aims to determine whether the device contains any information that are encrypted under a user-chosen passphrase and if so recover the password, and therefore the information. We assume passwords are a sequence of characters, including passphrases. Further, we assume that if the adversary can determine that encrypted information is present on the device, they pressure the user with this evidence and obtain the correct password. In other words, a successful scheme must provide plausible deniability and prevent an adversary from distinguishing between encrypted information and random data.

However, at the same time, plausibly-deniable encryption schemes, like ours, come with the drawback that the user cannot prove that a ciphertext *does not* store meaningful data [75]. An adversary—accepting to harm plausibly innocent users—might continue pressuring suspects to reveal passphrases even if it is likely that there is no encrypted data stored at all.

The input, i.e. the user password or passphrase, to our SLOTH schemes is typically not uniformly distributed. However, we assume that this input has "enough randomness" to make it hard for an attacker to guess and thus can be modelled as a min-entropy probability distribution:

Definition 1 (Min-Entropy Distribution [82])**.** *A probability distribution $\mathcal{X}$ has min-entropy (at least) m if for all a in the support of $\mathcal{X}$ and for random variable X drawn according to $\mathcal{X}$, $\textsc{Prob}(X = a) \leq 2^{-m}$.*

The adversary can also use the SE as a black box and can perform any operations via the available API (through oracle queries which we denote $\mathcal{O}_{SE}$). However, the adversary cannot extract information about the key material from the SE and they cannot clone the SE. Similar guarantees are given by Apple for their Secure Enclave [8] and Google for StrongBox implementations [58]. These claims are taken seriously by vendors: Apple and Google award up to \$250 000 [10] and \$1 000 000 [57], respectively, for the discovery of relevant vulnerabilities.

We introduce an abstract definition of an SE. The SE operates on a state ψ that can only be (meaningfully) accessed by it. In practice, many SEs have limited internal storage and store an authenticated ciphertext of their state on storage managed by the OS. This encryption is performed using a secret internal key. As the encrypted key blobs are stored outside the SE, this can allow an attacker to perform roll-back attacks which we discuss as a practical limitation in Section 5.3.3.

Definition 2 introduces a generic SE as there exist SEs with different capabilities. Later sections introduce more specialised definitions for SEs with various capabilities.

Definition 2 (Secure Element)**.** *A Secure Element SE operates on a hidden state $\psi \in \Psi$ using the algorithm SE.*$\textsc{Init}$ *and possible extension algorithms. It also has a timing function $T_{\textsc{Alg}}$ that maps each algorithm* $\textsc{Alg}$ *and its arguments to a wall time cost. SE.*$\textsc{Init} : \varnothing \to \Psi$ *initialises an empty state Ψ with $T_{SE.\textsc{Init}}() = 1$. Only the SE that initialised the state can operate on it with any of the extension algorithms.*

We generally assume the adversary only captures the device once, i.e. it is a single snapshot adversary. We believe that this limitation has little impact on the practicality of our scheme. If the user suspects that their phone has been tampered with, they can extract their data and perform a full device reset. Nevertheless, we propose an extra SE-backed layer of protection against multi-snapshot adversaries in Section 5.3.2.

We assume the adversary has a limited *wall time budget B* that is spent when performing oracle operations. The costs for each operation $\textsc{op}$ is defined by $\mathrm{T}_{\textsc{op}}$ and deduced from B before it is executed. We require that the adversary ends the experiment with a non-negative time budget $B \geq 0$. For this we build on the classic notion of probabilistic-polynomial time (PPT) algorithms [76, p. 46f]. We define for any PPT algorithm (including adversary and challenger):

Definition 3 (Wall Time Algorithm)**.** *A wall time (WT) algorithm is a PPT algorithm A with an initial wall time budget of $B_A \in \mathbb{N}$. During its execution it can perform the operations $\textsc{op}_i(p_1, p_2, \dots)$ given $\sum_i T_{\textsc{op}_i}(p_1, p_2, \dots) \leq B_A$.*

In comparison to standard security definitions, the strength is no longer supported by just asymptotic computational effort in $poly(\lambda)$, but also by a concrete wall time budget B. This is a strong property, as it is independent of advances in processing power or utilising more machines.

5.2.3 LongSloth key stretching for Android

LongSloth exclusively relies on symmetrical operations both outside and inside the Secure Element SE. This enables extremely efficient implementations on Android, but cannot be implemented on iOS (Section 5.2.4 for a variant compatible with iOS). Particularly, LongSloth operates on an SE equipped with HMAC support:

Definition 4 (SE with HMAC Support)**.** *A Secure Element with HMAC support SE-with-Hmac is an SE that has two extension algorithms:*

SE.HmacKeyGen : $\Psi \times H \to \Psi$ *generates a new secret key* k *and updates* $\psi \in \Psi$ *under handle* $h \in H$*:* $\psi.h \leftarrow k$ *requiring time* $T_{SE.HmacKeyGen}(\psi, h) = \Theta(1)$.

SE.Hmac : $\Psi \times H \times M \to \{0,1\}^\lambda$ *takes a message* $msg \in M = \{0,1\}^*$ *and a key handle* $h \in H$ *and outputs the corresponding* $HMAC(\psi.h, msg)$ *requiring time* $T_{SE.Hmac}(\psi, h, msg) = c_{Hmac} \cdot |msg| = \Theta(|msg|)$*, where* c_{Hmac} *is a device specific constant.*

Algorithm 3 describes the main operations of LongSloth. The procedure LongSloth.Derive (line 9) outputs a potentially existing key. During the pre-processing step (Figure 5.3), LongSloth expands a user password to a bit string ω_{pre}. This is achieved using the PwHash operation that reads a salt value from the local storage and then hashes the user password to a variable length output. The output length $l = |\omega_{pre}|$ is a configurable parameter to limit the guess rate based on the SE throughput rate. Next, ω_{pre} is computed by the SE-OP securely inside the SE with a key selected by the key handle h. This operation is an HMAC producing ω_{post} that is then hashed into the final key output $k = \text{Hkdf}(\omega_{post})$ using the hash-based key derivation function HKDF [82].

The generation of a new key k is done similarly. The procedure LongSloth.KeyGen (line 1) first initialises a new state π with the key handle h and a fresh random salt value. The SE then generates a new HMAC key under h. It returns the updated state and the key k by calling LongSloth.Derive.

Security intuition. LongSloth is designed to withstand an adversary with a wall time budget $B < 2^m \times l \times c_{\text{HMAC}}$, where m is the min-entropy of the password distribution. That is, the adversary would need to call 2^m times the operation SE.Hmac(x) with $|x| = l$ (Definition 3) to try all possible user passwords. Intuitively, the security of LongSloth relies on the observation that such an adversary is unable to distinguish a key k generated by LongSloth from a random bit string. Section 5.4 provides a formal security analysis based on the observation that HMAC is a PRF [22].

5.2.4 RainbowSloth key stretching for iOS

RainbowSloth relies on an SE equipped with support for Elliptic Curve Diffie-Hellman (ECDH) key exchanges (Definition 5). RainbowSloth is more complex than LongSloth but is compatible with both Android and iOS.

Algorithm 3 The LongSloth protocol with the security parameters l, λ freely chosen, but fixed.

procedure LongSloth.KeyGen(ψ, pw, h)
  $\pi \leftarrow \{\}$
  $\pi.h \leftarrow h$
  $\pi.salt \xleftarrow{\$} \{0,1\}^\lambda$
  $\psi \leftarrow$ SE.HmacKeyGen(ψ, h)
  $k \leftarrow$ LongSloth.Derive(π, ψ, pw)
  return (π, ψ, k)

procedure LongSloth.Derive(π, ψ, pw)
  $\omega_{pre} \leftarrow$ PwHash($\pi.salt, pw, l$)
  $\omega_{post} \leftarrow$ SE.Hmac($\psi, \pi.h, \omega_{pre}$)
  $k \leftarrow$ Hkdf(ω_{post})
  return k

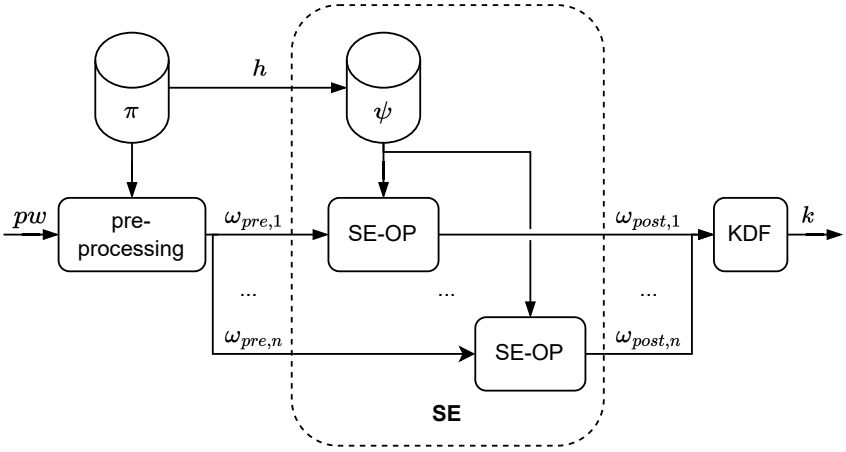

Figure 5.5: RainbowSloth scheme with a sequence of $\omega_{pre,i}$ inputs. The dashed area indicates the SE. Since there is no parallelism, all SE-OP are executed sequentially.

Definition 5 (SE with ECDH Support)**.** *A Secure Element with Elliptic Curve Diffie-Hellman support SE-with-Ecdh is a SE that has two extension algorithms:*

SE.EcdhPrivKeyGen : $\Psi \times H \to \Psi$ *generates a private EC key* k *and updates* $\psi \in \Psi$ *under handle* $h \in H$*:* $\psi.h \leftarrow k$ *requiring time* $T_{SE.EcdhPrivKeyGen}(\psi, h) = \Theta(1)$.

SE.Ecdh : $\Psi \times H \times pub \to \{0,1\}^\lambda$ *takes a EC public key* pub *and a key handle* $h \in H$ *and outputs the key exchange result* $Ecdh(\psi.h, pub)$ *requiring time* $T_{SE.Ecdh}(\psi, h, pub) = c_{Ecdh} = \Theta(1)$*, where* c_{Ecdh} *is a device specific constant.*

In contrast to LongSloth, RainbowSloth computes a sequence of fixed-sized public keys as its pre-secrets $\omega_{pre,i}$ (Figure 5.5). The processing speed of each ECDH operation is constant since the public key size is defined by the underlying elliptic curve. Instead of increasing the input length of each operation, RainbowSloth increases the required number of SE-OP executions to achieve a given throughput limit.

Algorithm 4 describes the main operations of RainbowSloth. The procedure RainbowSloth.Derive decrypts a potentially existing key. It first expands the user password into multiple

bit strings $\omega_{pre,i}$ with $i \in [1, n]$ calling PwHash. Those bit strings are then formatted into P-256 public keys (line 13). The SE uses the key handle h to select a private P-256 key and computes an ECDH operation with each $\omega_{pre,i}$ (line 14). Since the SE has no parallelism, these operations take place one after another. The resulting values $\omega_{post,i},\ i \in [1, n]$ are then jointly hashed into a final key k such that each contributes to all bits of k.

The generation of a new key k is done similarly. The procedure RainbowSloth.KeyGen (line 1) first initialises a new state π with the key handle h and a fresh random salt value. The SE then creates a new P-256 key under h. It returns all updated states and the key k by calling RainbowSloth.Derive as described above.

Algorithm 4 The RainbowSloth protocol with the security parameters n, λ freely chosen, but fixed.

```
 1: procedure RainbowSloth.KeyGen(ψ, pw, h)
 2:     π ← {}
 3:     π.h ← h
 4:     π.salt ←$ {0,1}^λ
 5:     ψ ← SE.EcdhPrivKeyGen(ψ, h)
 6:     k ← RainbowSlowth.Derive(π, ψ, pw)
 7:     return (π, ψ, k)
 8:
 9: procedure RainbowSloth.Derive(π, ψ, pw)
10:     l ← n × (256/8)
11:     ω_pre,1 ‖ ··· ‖ ω_pre,n ← PwHash(π.salt, pw, l)
12:     for i = 1 ... n do
13:         x ← ReHashToP256(ω_pre,i)
14:         ω_post,i ← SE.Ecdh(ψ, π.h, x)
15:     k ← Hkdf(ω_post,1 ‖ ··· ‖ ω_post,n)
16:     return k
```

Security intuition. RainbowSloth is designed to withstand an adversary with wall time budget $B < 2^m \times n \times c_{\text{ECDH}}$ (Definition 3), where m is the min-entropy of the password distribution. That is, the adversary would need to call $2^m \times n$ times SE.Ecdh to try all possible user passwords. Similarly to LongSloth, the security of RainbowSloth relies on the observation that the adversary is unable to distinguish a key k generated by RainbowSloth from a random bit string. Section 5.4 provides a formal security analysis assuming the SE runs the ECDH operations in a group where the generalised decisional Diffie-Hellman problem is hard [17].

Hashing into the P-256 Public Key Space. RainbowSloth requires an operation HashToP256 that maps any bit string $seed \in \{0,1\}^*$ to a valid P-256 public key. Each P-256 public key can be represented by 256 bits for its X-coordinate and a bit that determines the Y-coordinate [28]. However, not all possible 257-bit strings refer to valid public keys, as the key space only has size 2^{256}. We designed the ReHashToP256 algorithm and use it in our practical evaluation (Section 5.5).

The ReHashToP256 algorithm (Algorithm 5) considers the SEC-1 octal representation of P-256 curve points with point compression [28]. The representation for P-256 keys starts with a byte that is either `0x02` or `0x03` for the Y-coordinate information bit followed by 32 bytes for the X-coordinate. The algorithm repeatedly hashes the seed string and a counter to a candidate octet array. The first

octet is then set as `0x02` or `0x03` based on the parity of the original value of the first octet. We try to convert each candidate array to a P-256 public key using the Sec1OctetImportP256 algorithm [28, §2.3.4]. If it returns *invalid* ($\perp$) or the *point at infinity* ($\mathcal{O}$), the counter is incremented and the algorithm is repeated. Otherwise, a valid P-256 public key is returned (line 6). A similar approach is sketched in the Elligator paper [23, §1.4].

The runtime of this algorithm depends on finding a valid public key representation in one of its iterations. Since the output of the KDF can be considered pseudo-random, each iteration is independent and roughly half of the candidate octet arrays are invalid. We thus expect that the algorithm terminates with less than 10 iterations in $1 - 0.5^{10} \geq 99.9\%$ of all cases. Hashing with a counter instead of re-hashing the seed avoids falling into small loops that consist only of "bad seeds". The probability of the algorithm not terminating in polynomial-time is negligible.

Algorithm 5 The ReHashToP256 algorithm that maps any bit string $seed \in \{0,1\}^*$ to a valid P-256 public key.

1: $counter \leftarrow 0$
2: **while true do**
3: $\quad arr \leftarrow \text{KDF}(seed \parallel counter)[0:32]$
4: $\quad arr[0] = 0\text{x}02 \mid (arr[0] \;\&\; 0\text{x}01)$
5: $\quad x = \text{Sec1OctetImportP256}(arr)$
6: $\quad$ **if** $x \neq \perp$ **and** $x \neq \mathcal{O}$ **then return** x
7: $\quad counter \leftarrow counter + 1$

Alternatively, one might build an equivalent construction using the Elligator Squared (E^2) [132] technique. E^2 maps an elliptic curve point to a bit string that is indistinguishable from random. Such a bit string can then be mapped back to an elliptic curve point. While it can be computed efficiently [14], E^2 bit strings require more space. A more practical concern is that we are not aware of any freely available implementation. However, if deterministic runtime is required, E^2 can be used as a drop-in replacement for ReHashToP256.

5.3 The HiddenSloth deniable encryption scheme

Our generic methods to store and derive a key k can be used to build a deniable encryption (DE) scheme. We require a DE scheme to have three methods: (i) an Init method that initialises a new storage of a given maximum size, (ii) an Encrypt method that stores data encrypted with a given password, and (iii) a Decrypt method that retrieves and decrypts data from storage if there is any saved with the given password (and otherwise fails).

We present two variants of our deniable encryption scheme, 1S-HiddenSloth (Section 5.3.1) and MS-HiddenSloth (Section 5.3.2). 1S-HiddenSloth withstands an adversary capable of capturing a single snapshot of the user's device and MS-HiddenSloth withstands an adversary capable of capturing multiple snapshots at different points in time.

5.3.1 Single-snapshot variant

For most real-world scenarios, the single-snapshot adversary is the most realistic threat model. An adversary gaining access to a phone through finding or stealing it has no prior snapshot or knowledge

of the phone state, and hence cannot perform multi-snapshot attacks. Similarly, an adversary who confiscates a device can rarely do so covertly. When a user suspects an adversary captured a snapshot of their phone storage state, they can reset the phone and thus revert to a single-snapshot scenario. Nevertheless, there may be situations where an adversary can obtain multiple copies of the phone state over time; this is addressed in Section 5.3.2.

Algorithm 6 The *deniable encryption against single-snapshot* DESS protocol (1S-HiddenSloth) for maximum data size $s \leq 2^{31}$ and security parameter λ freely chosen, but fixed. The underlying Sloth protocol can be instantiated with either variant.

```
1:  procedure DESS.Init(ψ, h, s)
2:      pw ←$ {0,1}^λ
3:      ψ, π, k ← Sloth.KeyGen(ψ, pw, h)
4:      π ← DESS.Encrypt(π, ψ, pw, [])
5:      return (π, ψ)
6:
7:  procedure DESS.Encrypt(π, ψ, pw, data)
8:      k ← Sloth.derive(π, ψ, pw)
9:      x ← UInt32(|data|) ‖ data ‖ 0^(|π.data|−|data|−4)
10:     π.iv ←$ {0,1}^λ
11:     π.blob, π.tag ← AE.Enc(k, iv, x)
12:     return π
13:
14: procedure DESS.Decrypt(π, ψ, pw)
15:     k ← Sloth.derive(π, ψ, pw)
16:     x ← AE.Dec(k, π.iv, π.blob, π.tag)
17:     if x = ⊥ then return ⊥
18:     s′ ← UInt32(x[: 4])
19:     return x[4 : 4 + s′]
```

Algorithm 6 presents 1S-HiddenSloth, a single-snapshot resistant deniable encryption scheme. When calling the Init procedure, the algorithm allocates a storage file *blob* and fills it with random bytes to the given size (plus an allowance for overhead). This is implemented by encrypting a zeroed payload using a randomly chosen password (line 4). Upon encrypting new user data by calling the Encrypt procedure, the user provides a passphrase pw used to derive a cryptographic key k (using either LongSloth or RainbowSloth, see Sections 5.2.3–5.2.4). The payload is then prefixed with a 4-bytes integer representing its length and padded to the maximum size s. The resulting byte string is encrypted using the key k, for instance using AES-GCM. For the Decrypt operation, the key k is derived analogously and the algorithms attempt to decrypt the file. If this operation fails, an adversary will not be able to tell whether the passphrase was wrong or there was no previous call to Encrypt at all.

Besides our simple construction, Sloth can be used with existing designs that provide more features such as considerations of the flash storage layer and supporting both cover and multiple hidden volumes.

5.3.2 Multi-snapshot variant

In some scenarios the adversary might be able to access the storage state on multiple occasions. This may happen, e.g. if an app's data is backed up to the cloud to protect against device loss. Intuitively, 1S-HiddenSloth does not withstand such an adversary because changes in the ciphertext leak that the storage was overwritten between the device capture events.

MS-HiddenSloth overcomes this limitation and allows the adversary to capture the storage state π multiple times. However, our model restricts the adversary to only access the SE once they finally gain physical access to the device (recall Figure 5.4). That is, the adversary can perform SE calls only during the last device capture event. We believe this restriction is practical as smartphones are usually with the user and hence not likely available for covert access by the adversary. Also, if the adversary had such local privileged access multiple times they could simply install malware that records the keyboard and screen.

MS-HiddenSloth requires a SE that supports (unauthenticated) symmetric encryption:

Definition 6 (SE with Symmetric Encryption)**.** *A Secure Element with symmetric encryption support SE-with-SymmEnc is an SE that has three extension algorithms:*

- $SE.\textsc{SymmKeyGen} : \Psi \times H \to \Psi$ *generates a new secret key* k *and updates* $\psi \in \Psi$ *under handle* $h \in H$*:* $\psi.h \leftarrow k$ *requiring time* $T_{SE.\textsc{SymmKeyGen}}(\psi, h) = \Theta(1)$.
- $SE.\textsc{SymmEnc} : \Psi \times H \times IV \times M \to C$ *takes an initialisation vector* $iv \in IV$ *and a key handle* $h \in H$ *and outputs the ciphertext* $\textsc{SymmEnc.Encrypt}(\psi.h, iv, m) = c \in C$ *for the message* $msg \in M$ *requiring time* $T_{SE.\textsc{SymmEnc}}(\psi, h, msg) = \Theta(|msg|)$.
- $SE.\textsc{SymmDec} : \Psi \times H \times IV \times C \to M$ *takes an initialisation vector* $iv \in IV$ *and a key handle* $h \in H$ *and outputs the message* $\textsc{SymmEnc.Decrypt}(\psi.h, iv, c) = msg \in M$ *for the ciphertext* $c \in C$ *requiring time* $T_{SE.\textsc{SymmEnc}}(\psi, h, c) = \Theta(|c|)$.

To protect against a multi-snapshot adversary MS-HiddenSloth wraps the storage in another layer of encryption guarded by an additional symmetric key k_{SE} held in the SE's secure state under handle h'. The outer layer is periodically re-encrypted using a new temporary key tk, which in turn is re-encrypted with a fresh k_{SE}. The re-encryption process (DEMS.Ratchet) should happen at least as often as the adversary has the opportunity to access the stored data. For instance, for a backed-up app, this would happen after every upload operation. In other cases, every app restart could be used as a trigger event. An important design feature of MS-HiddenSloth is that it does not require the user's password to execute the re-encryption process; that is, the procedure DEMS.Ratchet can be executed entirely in the background without user interaction.

MS-HiddenSloth algorithm. Algorithm 7 presents all the operations of MS-HiddenSloth. The initial state is initialised similarly to 1S-HiddenSloth but with the creation of an additional SE secret key (k_{SE}) associated with the handle h' (line 4). To encrypt new data, MS-HiddenSloth first encrypts them as in 1S-HiddenSloth; it then generates an ephemeral key tk that is used to re-encrypt the data (line 11). This ephemeral key is finally encrypted using the SE's secret key generated at line 4 and stored in the user space π. To decrypt existing data, MS-HiddenSloth first decrypts the ephemeral key tk (line 19); it then uses that key to decrypt the outer encryption layer (line 20) and finally decrypt the data (line 21).

Algorithm 7 The *deniable encryption against multi-snapshot* DEMS protocol (HiddenSloth) for maximum data size $s \leq 2^{31}$ and security parameter λ freely chosen, but fixed. The underlying Sloth protocol can be instantiated with either variant.

procedure DEMS.Init(ψ, h, s)
  $\pi, \psi, k \leftarrow$ DESS.Init($\psi, h \parallel 0, s$)
  $\pi.h' \leftarrow h \parallel 1$
  $\psi \leftarrow$ SE.SymmKeyGen($\psi, \pi.h'$)
  $pw \xleftarrow{\$} \{0,1\}^\lambda$
  $\pi \leftarrow$ DEMS.Encrypt($\pi, \psi, pw, [\,]$)
  return (π, ψ)

procedure DEMS.Encrypt($\pi, \psi, pw, data$)
  $\pi \leftarrow$ DESS.Encrypt($\pi, \psi, pw, data$)
  $\pi.tk \leftarrow$ AE.KeyGen(); $\pi.tiv \xleftarrow{\$} \{0,1\}^\lambda$
  $\pi.blob, \pi.ttag \leftarrow$ AE.Enc($\pi.tk, \pi.tiv, \pi.blob$)
  for $\mathcal{K} \in \{iv, tag, tk, tiv, ttag\}$ **do**
    $\pi.\mathcal{K} \leftarrow$ SE.SymmEnc($\psi, \pi.h', \pi.\mathcal{K}$)
  return π

procedure DEMS.Decrypt(π, ψ, pw)
  for $\mathcal{K} \in \{iv, tag, tk, tiv, ttag\}$ **do**
    $\pi.\mathcal{K} \leftarrow$ SE.SymmDec($\psi, \pi.h', \pi.\mathcal{K}$)
  $\pi.blob \leftarrow$ AE.Dec($\pi.tk, \pi.tiv, \pi.blob, \pi.ttag$)
  return DESS.Decrypt(π, ψ, pw)

procedure DEMS.Ratchet(π, ψ)
  for $\mathcal{K} \in \{iv, tag, tk, tiv, ttag\}$ **do**
    $\pi.\mathcal{K} \leftarrow$ SE.SymmDec($\psi, \pi.h', \pi.\mathcal{K}$)
  $\pi.blob \leftarrow$ AE.Dec($\pi.tk, \pi.tiv, \pi.blob, \pi.ttag$)
  $\pi.tk \leftarrow$ AE.KeyGen(); $\pi.tiv \xleftarrow{\$} \{0,1\}^\lambda$
  $\pi.blob, \pi.ttag \leftarrow$ AE.Enc($\pi.tk, \pi.tiv, \pi.blob$)
  $\psi \leftarrow$ SE.SymmKeyGen($\psi, \pi.h'$)
  for $\mathcal{K} \in \{iv, tag, tk, tiv, ttag\}$ **do**
    $\pi.\mathcal{K} \leftarrow$ SE.SymmEnc($\psi, \pi.h', \pi.\mathcal{K}$)
  return (π, ψ)

The re-encryption process (ratchet) works similarly to decryption followed by an encryption operation. It firsts retrieves the temporary key *tk* and uses it to decrypt the outer encryption layer. It then generates a new temporary key that is used to re-encrypt the data and thus create a new outer encryption layer; the new temporary key is then encrypted by the SE and persisted in the user space. For performance reasons, MS-HiddenSloth does not perform the re-encryption process over the actual data $\pi.blob$ inside the SE; it instead re-encrypts the data with a temporary key $\pi.tk$. This temporary key $\pi.tk$ and its related fields $\pi.tiv$, $\pi.ttag$ are freshly generated at every encryption (line 11) and ratchet step (line 27); they are only stored encrypted with the SE's secret key generated at line 4. The indirection via $\pi.tk$ allows to leverage the faster AES engine on the main CPU without sacrificing security.

Security intuition. This extra encryption layer is sufficient to withstand a multi-snapshot adversary, as the adversary will always encounter a random-looking ciphertext. When the adversary finally gains access to the SE, they can only reverse it for the last ciphertext, but not for any beforehand. A similar technique of adding a reversible encryption layer is not possible in existing schemes without an SE as the encryption key would be part of the captured storage state. An alternative would be decrypting and re-encrypting the user data at every event. This would however be impractical as it necessitates the user to input their passphrase for each of these events.

5.3.3 Practical implementation considerations

Sloth is generally more practical than other schemes that rely on compute and memory-hard functions. This is because their parameters have to be chosen from the perspective of an attacker with access to a large parallel cluster of machines that are more powerful than a smartphone.

For the regular Sloth key stretching schemes and 1S-HiddenSloth, the implementation must not use a file system or underlying storage technology that allows an attacker to discover whether and when a file was overridden. However, since the ratchet steps for MS-HiddenSloth are assumed to be predictable from the adversary's perspective, its security guarantees hold for any form of storage.

The actual implementation into a production app also needs to consider the allocated space for the deniable encryption since it should always be the same size regardless of the actual usage. Developers have to weigh the costs of having a large encrypted file for non-users against the storage requirements of its active users. Since MS-HiddenSloth regularly re-encrypts all its data, the performance and energy impact need to be considered as well. Finally, developers must take care that the deniable parts of the application do not leave any other traces in the form of crash reports and logging output.

In addition to application-specific storage, the details of how the SE is implemented and stores its state are important as well. In particular, the limited on-chip storage means that SEs typically persist their state as encrypted key blobs and metadata using the main device storage. While the used techniques generally provide strong confidentiality and integrity, they often only provide coarse roll-back protection. For instance, on Android, only OS updates cause the StrongBox roll-back protection counter to increment. As such, Android devices are vulnerable to roll-back attacks where the adversary can replace the encrypted key blobs[2] with older versions. The iOS documentation is inconclusive regarding roll-back protection for stored secrets on iPhones and iPads.

[2]Typically stored in `/data/misc/keystore/persistent.sqlite`

For our HiddenSloth scheme which relies on ratcheting keys, the missing roll-back protection can allow an adversary to detect changes if they both capture the encrypted key blobs and Sloth ciphertexts before and after a suspected usage event *and* later gain oracle access to the SE. However, we note that the encrypted key blobs are typically not accessible by application processes and they are never backed up by the system. Therefore, the ratcheting mechanism still provides value for scenarios where an adversary gains access to online backups that inadvertently include Sloth ciphertexts.

5.4 Security analysis

We formally capture the security of our Sloth schemes using experiments between a challenger $\mathcal{C}$ and a wall-time bounded (WT) adversary $\mathcal{A}$ (Definition 3).

5.4.1 Security of the key stretching schemes

We prove the security of LongSloth and RainbowSloth. For this we formally define an SE-backed key stretching scheme and and experiments for key stretching indistinguishability and hardness. The success of a WT adversary is controlled by the time required by the SE to execute operations $T_{\text{SE-OP}}$ (and the Sloth parameters).

Definition 7. *A key stretching scheme Ξ for fixed security parameter λ consists of two algorithms:*

- $\textsc{KeyGen} : \Psi \times \mathcal{P} \times H \to \Pi \times \Psi \times \{0,1\}^\lambda$ *takes an SE state, a password, and a key handle and returns a new storage state, an updated SE state, and the derived key.*
- $\textsc{Derive} : \Pi \times \Psi \times \mathcal{P} \to \{0,1\}^\lambda$ *takes a storage state, SE state, and a password to derive a key $k \in \{0,1\}^\lambda$.*

We require for correctness that after the initialisation operation $(\pi, \psi, k) \leftarrow \textsc{KeyGen}(\psi', pw, h)$ and all subsequent $\textsc{Derive}(\pi, \psi, pw)$ executions return the same k.

The adversary must not learn any information about either the password or the generated key. We propose an experiment where the adversary is required to distinguish between a truly random bit string and a key generated from a password. If an WT adversary fails to do so better than with negligible probability, even when given access to the stored information and oracle access to the SE, we say that the key stretching scheme is indistinguishable.

Definition 8 (Key Stretching Indistinguishability Experiment)**.** *Let $\mathcal{P}$ be a probability distribution with min-entropy m. Let $\mathcal{A}$ be a WT adversary with wall time budget B and $\mathcal{C}$ a WT challenger. Let Ξ be a key stretching scheme. Then the key stretching indistinguishability experiment $\textsc{KeyInd}_{\mathcal{A}_\Xi}(\lambda, \mathcal{P})$ is defined as follows:*

1. *The adversary $\mathcal{A}$ provides the challenger $\mathcal{C}$ with a password $pw \xleftarrow{\$} \mathcal{P}$.*
2. *$\mathcal{C}$ randomly samples $b \xleftarrow{\$} \{0,1\}$. Let ψ be freshly initialised and h an arbitrary, but fixed key handle.*
3. *$\mathcal{C}$ computes $\pi, \psi, k_0 \leftarrow \Xi.\textsc{KeyGen}(\psi, pw, h)$ under λ and samples $k_1 \leftarrow \{0,1\}^\lambda$.*
4. *The adversary $\mathcal{A}$ receives (π, ψ, k_b).*
5. *$\mathcal{A}$ receives oracle access $\mathcal{O}_{SE}$ (WT conditions).*

6. *$\mathcal{A}$ outputs a bit b' and wins iff $b = b'$.*
7. *The experiment returns 1 iff $\mathcal{A}$ wins, otherwise 0.*

Definition 9 (Key Stretching Indistinguishability)**.** *A key stretching scheme Ξ is m-entropy indistinguishable if for all WT adversaries $\mathcal{A}$, there is a negligible function* NEGL*:*

$$Pr[\textsc{KeyInd}_{\mathcal{A},\Xi}(\lambda, \mathcal{P}) = 1] \leq \frac{1}{2} + \textsc{negl}(\lambda),$$

where $\mathcal{P}$ is a probability distribution with min-entropy m.

The LongSloth and RainbowSloth schemes respectively described in Section 5.2.3 and Section 5.2.4 fulfil these definitions as per the following theorems.

Theorem 10 (LongSloth Indistinguishability)**.** *Let $\mathcal{P}$ be a random distribution with min-entropy m. The key stretching scheme LongSloth is m-entropy indistinguishable.*

Proof. We present a full proof in Appendix B.1.1. □

Theorem 11 (RainbowSloth Indistinguishability)**.** *Let $\mathcal{P}$ be a random distribution with min-entropy m. The key stretching scheme RainbowSloth is m-entropy indistinguishable.*

Proof. We present a full proof in Appendix B.2.1. □

In addition to the theoretical result above, we are interested in the practical success rate of a brute-force attacker, i.e. given k find pw with wall time budget B.

Definition 12 (Key Stretching Hardness Experiment)**.** *Let $\mathcal{P}$ be a probability distribution with min-entropy m. Let $\mathcal{A}$ be a WT adversary with wall time budget B and $\mathcal{C}$ a WT challenger. Let Ξ be a key stretching scheme. Then the key stretching hardness experiment $\textsc{KeyHard}_{\mathcal{A}_\Xi}(\lambda, \mathcal{P})$ is defined below:*

1. *$\mathcal{C}$ samples a password $pw \xleftarrow{\$} \mathcal{P}$. Let ψ be freshly initialised and h an arbitrary (but fixed) key handle.*
2. *$\mathcal{C}$ computes $k = \Xi.\textsc{KeyGen}(\psi, pw, h)$.*
3. *$\mathcal{A}$ receives (π, ψ, k).*
4. *$\mathcal{A}$ receives oracle access $\mathcal{O}_{SE}$ (WT conditions).*
5. *$\mathcal{A}$ outputs a pw' and wins iff $k = \Xi.\textsc{Derive}(\psi, pw', h)$.*
6. *The experiment returns 1 iff $\mathcal{A}$ wins, otherwise 0.*

Definition 13 (Key Stretching Hardness)**.** *A key stretching scheme Ξ is σ-hard if for all WT adversaries $\mathcal{A}$ with wall-time budget B,*

$$Pr[\textsc{KeyHard}_{\mathcal{A},\Xi}(\lambda, \mathcal{P}) = 1] \leq \frac{B}{\sigma \cdot 2^m},$$

where $\mathcal{P}$ is a probability distribution with min-entropy m.

Theorem 14 (LongSloth Hardness)**.** *Let $\mathcal{P}$ be a random distribution with min-entropy m. Then the key stretching scheme LongSloth with parameter l is $(l \cdot c_{\textsc{Hmac}})$-hard.*

Proof. We present a full proof in Appendix B.1.2. □

Theorem 15 (RainbowSloth Hardness)**.** *Let $\mathcal{P}$ be a random distribution with min-entropy m. Then the key stretching scheme RainbowSloth with parameter n is $(n \cdot c_{ECDH})$-hard.*

Proof. We present a full proof in Appendix B.2.2. □

5.4.2 Security of the deniable encryption schemes

Similarly to key stretching, we discuss the security of deniable encryption by giving formal definition, describing security experiments, and then showing that an adversary has at most a negligible advantage. We only discuss the multi-snapshot case and prove the security of MS-HiddenSloth. A similar (and simpler) reasoning applies to 1S-HiddenSloth.

Definition 16 (SE with MS Deniable Encryption Support)**.** *A SE-backed deniable encryption scheme Δ for fixed security parameter λ and maximum data size $s \in \mathbb{N}$ consists of three algorithms:*

- *Init* : $\Psi \times H \times \mathbb{N} \to \Pi \times \Psi$ *takes a SE state, a key handle, and storage size s and returns a new storage state and SE state.*
- *Encrypt* : $\Pi \times \Psi \times \mathcal{P} \times \{0,1\}^s \to \Pi$ *updates a storage state for given SE state and a password so that it now stores the given data.*
- *Decrypt* : $\Pi \times \Psi \times \mathcal{P} \to \{0,1\}^s \cup \{\perp\}$ *tries to decrypt from the storage state given the password. If successful, the previously stored data is returned, otherwise $\perp$.*
- *Ratchet* : $\Pi \times \Psi \to \Pi \times \Psi$ *updates the storage and SE state by decrypting and re-encrypting the stored data.*

Definition 17 (MS Deniable Encryption Indistinguishability Experiment)**.** *Let λ be a fixed security parameter and s the maximum data size. Let $\mathcal{P}$ be a probability distribution with min-entropy m. Let $\mathcal{A}$ be a WT adversary with wall time budget B and $\mathcal{C}$ a WT challenger. Let Δ be a multi-snapshot deniable encryption scheme. Then the multi-snapshot deniable encryption indistinguishability experiment DE-MS-Ind$_{\mathcal{A},\Delta}(\lambda, \mathcal{P})$ is defined as follows:*

1. *The challenger C randomly samples $b \xleftarrow{\$} \{0,1\}$ and $pw \xleftarrow{\$} \mathcal{P}$. Let ψ_0, ψ_1 be freshly initialised and h an arbitrary, but fixed key handle.*
2. *C computes $\pi_0, \psi_0 \leftarrow \Delta.\textit{Init}(\psi_0, h, s)$ and $\pi_1, \psi_1 \leftarrow \Delta.\textit{Init}(\psi_1, h, s)$ under λ.*
3. *A sends any $\tilde{b} \in \{0,1\}$ and $msg \in \{0,1\}^s$ to C.*
4. *C executes $\pi_{\tilde{b}} \leftarrow \Delta_{MS}.\textit{Encrypt}(\pi_{\tilde{b}}, \psi_{\tilde{b}}, pw, msg)$, then C sends $\pi_{\tilde{b}}$ to A.*
5. *A can repeat execution from step 3 several times (under WT conditions).*
6. *C randomly samples $b \xleftarrow{\$} \{0,1\}$, computes $\pi_b, \psi_b \leftarrow \Delta.\textit{Ratchet}(\pi_b, \psi_b)$, and provides A with (π_b, ψ_b).*
7. *A receives oracle access $\mathcal{O}_{SE}$ under the WT conditions.*
8. *A outputs a bit b' and wins iff $b = b'$.*
9. *The experiment returns 1 iff A wins, otherwise 0.*

Definition 18 (MS Deniable Encryption Indistinguishability)**.** *A multi-snapshot deniable encryption scheme Δ is B-time m-entropy secure if for all WT adversaries $\mathcal{A}$ with time budget B, there is a function* NEGL *such that*

$$Pr[\text{DE-MS-IND}_{\mathcal{A},\Delta}(\lambda, \mathcal{P}) = 1] \leq \frac{1}{2} + \text{NEGL}(\lambda),$$

where $\mathcal{P}$ is a probability distribution with min-entropy m.

Theorem 19. *(MS-HiddenSloth Indistinguishability) The multi-snapshot deniable encryption scheme MS-HiddenSloth is B-time m-entropy secure.*

Proof. We present a full proof in Appendix B.3.1. □

Similarly to the key stretching scheme, we analyse the practical success rate of a brute-force attacker.

Definition 20 (Deniable Encryption Hardness Experiment)**.** *Let $\mathcal{P}$ be a probability distribution with min-entropy m. Let $\mathcal{A}$ be a WT adversary with wall time budget B and $\mathcal{C}$ a WT challenger. Let Δ be a key stretching scheme. Then the deniable encryption hardness experiment $\text{DeHard}_{\mathcal{A}_\Delta}(\lambda, \mathcal{P})$ is defined as follows:*

1. *$\mathcal{A}$ provides $\mathcal{C}$ with data with $|data| > 0$.*
2. *$\mathcal{C}$ samples a password $pw \xleftarrow{\$} \mathcal{P}$. Let ψ be freshly initialised and h an arbitrary (but fixed) key handle. Let $\pi, \psi = \Delta.\text{Init}(\psi, h, |data|)$.*
3. *$\mathcal{C}$ computes $\pi = \Delta.\text{Encrypt}(\pi, \psi, pw, data)$.*
4. *$\mathcal{A}$ receives (π, ψ).*
5. *$\mathcal{A}$ receives oracle access $\mathcal{O}_{SE}$ (WT conditions).*
6. *$\mathcal{A}$ outputs a pw' and wins iff $data = \Delta.\text{Decrypt}(\pi, \psi, pw')$.*
7. *The experiment returns 1 iff $\mathcal{A}$ wins, otherwise 0.*

Definition 21 (Deniable Encryption Hardness)**.** *A deniable encryption scheme Δ is σ-hard if for all WT adversaries $\mathcal{A}$ with wall-time budget B,*

$$Pr[\text{DeHard}_{\mathcal{A},\Delta}(\lambda, \mathcal{P}) = 1] \leq \frac{B}{\sigma \cdot 2^m},$$

where $\mathcal{P}$ is a probability distribution with min-entropy m.

Theorem 22 (MS-HiddenSloth Hardness)**.** *Let $\mathcal{P}$ be a random distribution with min-entropy m. Then the deniable encryption scheme MS-HiddenSloth instantiated with a σ-hard key stretching scheme is σ-hard.*

Proof. We present a full proof in Appendix B.3.2. □

5.5 Evaluation

We first measure the performance of SEs in Android and iOS devices (Section 5.5.1). These numbers will then guide the choice of practical parameters for our schemes (Section 5.5.2). With these

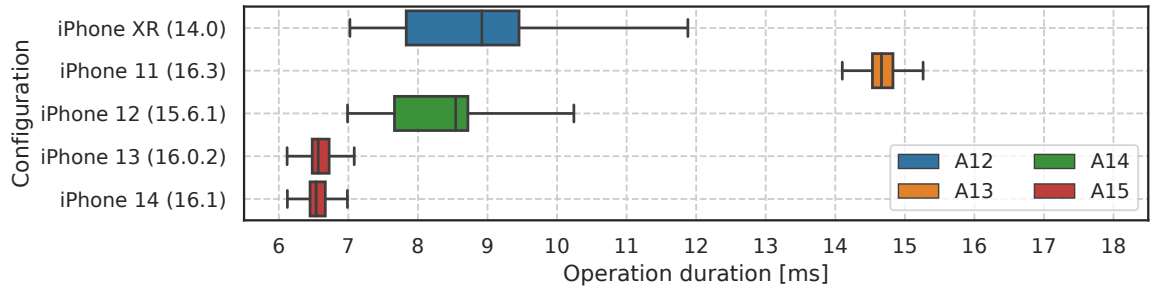

Figure 5.6: The duration of the ECDH operation on iOS for different phones (iOS version in brackets). The colours indicate the chip generation used in the respective phone.

parameters, we test full implementations of LongSloth and RainbowSloth on Android and iOS devices (Section 5.5.3). We also analyse the Ratchet operation of HiddenSloth (Section 5.5.4).

5.5.1 Performance of Secure Element operations

For iOS devices we measure the duration of the Secure Enclave's ECDH operations on the recent iPhones from XR to 14. These cover the Apple chips A12, A13, A14, and A15. We wrote a benchmark app that first creates a secret P-256 key within the SE. It then creates a new random public key and performs ECDH with the private key within the SE. We repeat the ECDH step 1 000 times with the new public keys and measure the elapsed time. The results (Figure 5.6) show that similar chips have similar run times, e.g. A15 for iPhone 13 and iPhone 14. All measurements are between 6 ms and 16 ms with little variance per model. Surprisingly, the A13 chip has lower throughput than its predecessor. However, A13 is also the first one with a mathematically verified public key implementation [8] which might point to a difference in the underlying algorithm.

For Android devices, we measure the duration of HMAC executions for inputs of varying lengths on multiple devices. We wrote a benchmark tool that first creates a secret key within the SE. It then generates random byte arrays as inputs, passes these as input to the SE, and receives the computed HMAC value as a result. We repeat this step 10 times for each device and input length to measure the elapsed time. The results are shown in Figure 5.7. Intuitively, the measured time increases with input length in an almost linear manner. There are minor inflection points that differ between the devices. In our experiments, the Samsung phones also have a 10× higher bandwidth compared to the Google devices. For our parameter choice, this will result in a larger l value for those devices. Notably, for the same device and input length, the variance is very small. Once the parameters are established for each device, their impact is predictable and dependable.

5.5.2 Choosing Sloth parameters

For our evaluation, we target a security level of $T_{total} = 100$ years (Table 5.3). We treat the acceptable password complexity, such as the alphabet size and the number of characters, as input parameters. Other than classic alphanumerical passwords[3], we also consider passphrases that are based on the EFF word list [26] that contains 7776 words and PINs consisting of only digits. Let $|A|$ be the alphabet

[3]Alphanumerical passwords can contain the case-sensitive letters `a-zA-Z` and the digits `0-9`. Hence, $|A| = 76$.

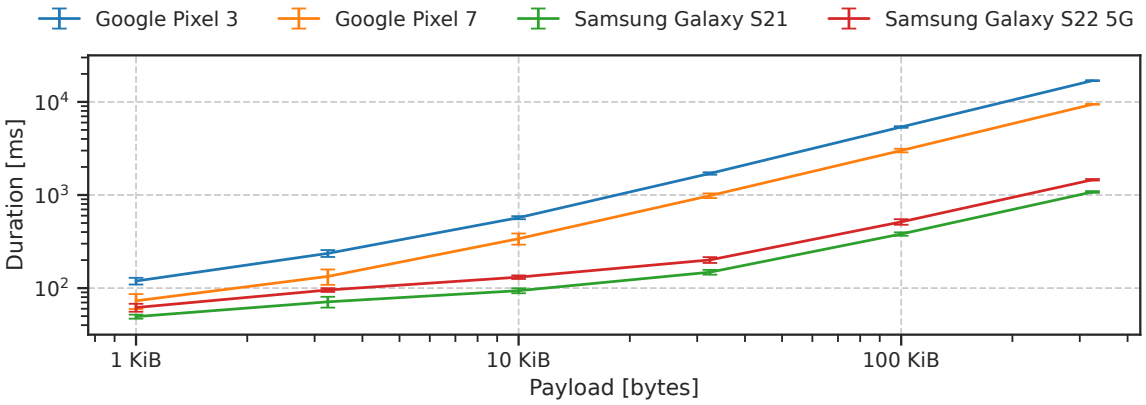

Figure 5.7: The duration of a HMAC operation on Android within the StrongBox supported chip for different payload sizes and phones. Both axes are log-scale.

	Entropy	50 years	100 years
WordList (3 words) = **c1**	38.8	3.4 ms	6.7 ms
WordList (4 words)	51.7	0.1 ms	0.1 ms
AlphaNumerical (5 characters)	29.8	1.7 s	3.4 s
AlphaNumerical (6 characters) = **c2**	35.7	27.8 ms	55.5 ms
AlphaNumerical (7 characters)	41.7	0.4 ms	0.9 ms
PIN (9 digits)	29.9	1.6 s	3.2 s

Table 5.3: Overview of password configurations, their entropy in bits, and the required $t_{individual}$ for achieving given security margins.

	3 words $t_{c1} = 67\,\text{ms}$	6 characters $t_{c2} = 555\,\text{ms}$
LongSloth (parameter: **l**)		
Google Pixel 3	1 500	11 600
Google Pixel 7	4 500	24 200
Samsung Galaxy S21	10 700	178 000
Samsung Galaxy S22	2 200	145 100
RainbowSloth (parameter: **n**)		
iPhone 11	6	43
iPhone 12	10	76
iPhone 13	12	95

Table 5.4: Smallest possible parameter choices for given password configuration and its $t_{individual}$ augmented by a safety factor of 10×.

size and $|pw|$ the password length, then the entropy for a configuration is $e = \log_2(|A|^{|pw|})$. We want that an attacker's worst-case to match the targeted security level, i.e. if they brute-force the entire space, then they require at least T_{total}. Let T_{total} be the targeted security level, then for a password configuration with entropy e each password verification should take at least $t_{individual} = T_{total} \times 2^{-e}$. We provide sample numbers for different configurations in Table 5.3.

The password configurations that we show in this table are shorter than those used in web applications nowadays. This is because for Sloth the parameter choice does not need to conservatively assume an attacker with highly parallel computing resources. Short passwords, and in particular passphrases generated from word lists, are more memorable and can be generated by the application for the user — and thus avoid the problem of users picking weak passwords or reusing the same one for different services.

In our main evaluation (Section 5.5.3) we will examine two configurations: 3-word passphrases from the EFF word list ($c1$) and 6-character alphanumerical passwords ($c2$). In both cases, we multiply the minimal $t_{individual}$ (Table 5.3) by a **safety factor of ×10** for conservative parameter choice. Therefore: $t_{c1} = 67\,\text{ms}$ and $t_{c2} = 555\,\text{ms}$. This safety factor can account for the attacker over-clocking the SE and overhead by the operating system when communicating with the chip that an attacker might be able to "optimise away".

With the required minimum times t_{c1} and t_{c2} for the individual operations, we determine l for LongSloth on Android and n for RainbowSloth on iOS. On Android, we use the measurements from Figure 5.7 to fit a second-degree polynomial where we set the y-values to the smallest measurement for a given size minus 2 standard deviations. We then pick the l value at the intersection with the desired duration and round to the next multiple of 100. On iOS we pick the 10th percentile value t_{p10} of our measurements as a conservative worst-case (i.e. fastest) duration and then compute $n_c = \lceil \frac{t_c}{t_{p10}} \rceil$ for $c \in \{c1, c2\}$. The resulting values are summarised in Table 5.4. If an app cannot find existing parameters for a new device type, it can perform a similar method on its first start to self-calibrate.

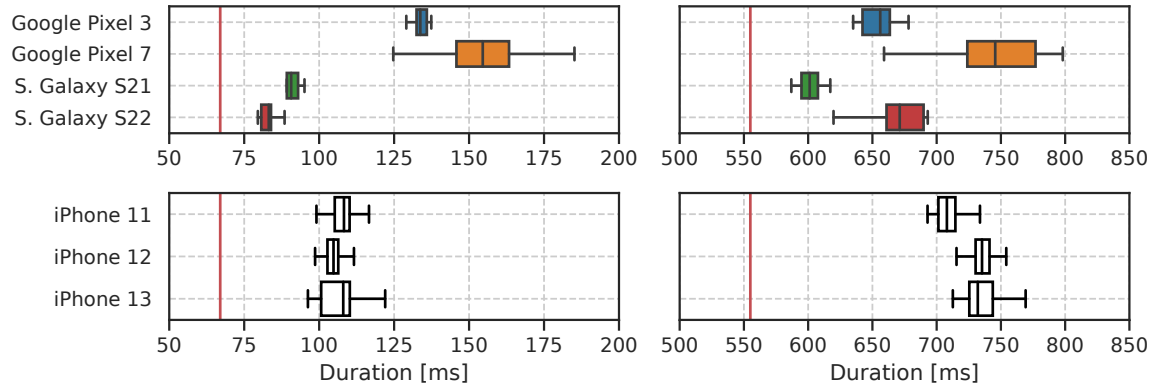

Figure 5.8: The duration of the LongSloth.Derive operation on Android (top) RainbowSloth.Derive operation on iOS (bottom). For both we evaluated the configurations $c1$ (left) and $c2$ (right) from Table 5.4 for different phones. The red lines indicate the threshold times t_{c1} and t_{c2}.

5.5.3 LongSloth and RainbowSloth

We implemented LongSloth on Android and RainbowSloth on iOS. All code including documentation and analysis scripts will be made available under an open-source license. Where possible we use the existing cryptography APIs of the platform, with AES-GCM for symmetric authenticated encryption, and HKDF-SHA256 as the KDF. For PwHash we use the third-party LibSodium library which is available on both platforms and implements the memory-hard password hashing algorithm Argon2 [24]. For the evaluation, we choose the recommended OWASP parameters for Argon2id with 19 MiB of memory and an iteration count of 2 [102]. Since one could opt for another PwHash implementation, we exclude its runtime (50 ms) from our results for LongSloth and RainbowSloth.

For our evaluation, we are interested in the duration of the Sloth.derive operations, as these determine the time costs for an attacker. We use the parameters as chosen in Section 5.5.2 including the safety factor and execute each operation 10 times. The results are shown in Figure 5.8 for LongSloth on Android RainbowSloth on iOS. In all cases, the measured total durations comfortably exceed the threshold times t_{c1} and t_{c2}. This confirms that with our parameter choice, the algorithm meets its minimum timing promise and hence key stretching security. The variance for the individual configurations is small which can allow for reducing the safety factor.

5.5.4 HiddenSloth

For the deniable encryption scheme HiddenSloth we evaluate its Ratchet methods for varying maximum data sizes s. This parameter s spans from a small storage size that might wrap text-only configurations (1 MiB) to larger ones that can store long chat histories including media (100 MiB). In these experiments, we use the faster t_{c1} for the underlying Sloth schemes. As HiddenSloth requires an SE-with-SymmEnc, we evaluate it only on Android devices. Our results are shown in Figure 5.9. The measured durations are similar for all test devices and range from less than 1 s for 1 MiB to about 10 s for storage with 100 MiB capacity. We note that the Ratchet step does not

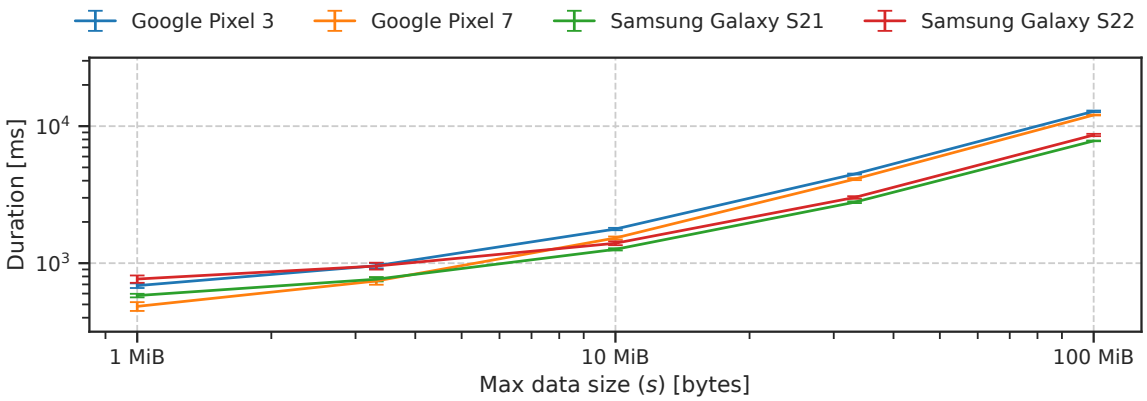

Figure 5.9: Duration of the HiddenSloth.Ratchet step for different phones (various max size s). Both axes are log-scale.

require any user interaction and thus can be executed in the background. As such, the multi-second duration has no user-visible impact and we suggest scheduling it when the device is idle and charging.

5.6 Summary

In this chapter we presented the LongSloth and RainbowSloth key stretching schemes that use the SE to provide strict time guarantees. This means that adversaries cannot speed up brute-force attacks by using more computers. Notably, Sloth can be used today on modern smartphones as a drop-in replacement for other password-based key derivation functions without requiring modifications to software or hardware. Based on the Sloth key stretching schemes we constructed the plausibly-deniable encrypted storage protocol HiddenSloth. On Android we can extend HiddenSloth so that it additionally supports resistance against multi-snapshot adversaries through a ratchet mechanism based on a SE-backed key.

In our survey of SE availability we showed that support is widespread and becoming standard on both modern Android and iOS devices. Our implementations of LongSloth, RainbowSloth, and HiddenSloth demonstrate that they are practical on these devices without any additional changes. In our evaluation we show that both key stretching schemes—and hence also HiddenSloth—work well with short, memorable passphrases that consist of only few words. This is important as it allows the application to provide randomly-generated passphrases to avoid weak user-chosen passphrases. Finally, we formalised the security of SE-based operations and used this to prove the security of our Sloth schemes.

In the next chapter we present CoverDrop, a system where whistleblowers can establish contact with journalists in a secure and anonymous way. CoverDrop is integrated as an additional feature in the regular news reader app, allowing all users to form a common anonymity set. Importantly, a source must be able to later deny that they used CoverDrop to send messages—even if an adversary captures their smartphone. Therefore, the active conversations and key material are stored using a plausibly-deniable storage protocol like HiddenSloth.

Chapter 6

Real-world implementation of the CoverDrop anonymous messaging system

In this chapter we discuss CoverDrop as a case study of a practical mobile application with strong metadata privacy. CoverDrop allows sources to reach out to journalists securely and anonymously through a dedicated feature embedded in a regular news reader app. CoverDrop focuses on the initial contact, where both parties build trust and establish rapport. The scope and direction of this project are the result of an extensive requirements gathering process (Section 6.1.3) with investigative journalists and security engineers.

Whistleblowers are always at risk of being investigated by the organisation that they expose. Therefore, CoverDrop must not only provide strong confidentiality and metadata privacy of the network traffic; in addition it must also provide plausible deniability if a suspected whistleblower is forced to hand-over their phone or coerced into entering their passphrase. This motivates the integration of a plausibly-deniable storage, e.g. HiddenSloth (Section 5.3), into the news app.

The original paper "CoverDrop: Blowing the Whistle Through A News App" [2] introduced our solution for the first time. Mansoor, Diana, Alastair, and Ross came up with the initial research direction and conducted the workshops that shaped the main set of requirements. Afterwards, all co-authors have been working on the system design together. I was the main contributor regarding the mobile aspects and the plausibly-deniable storage using the SE which would later turn into Sloth (Chapter 5) as a project on its own. For the practical evaluation, I implemented an end-to-end prototype that included a newsreader app and the backend services. Mansoor implemented a proof-of-concept of a CoverNode that uses Intel SGX for its most-sensitive operations.

After we presented our work at the Privacy Enhancing Technologies Symposium, we have established a collaboration with a large British news organisation with the goal of implementing and deploying CoverDrop in the news organisation's main news reader app which supports millions of daily users. The work on the production-grade implementation itself revealed short-comings that are typically out-of-scope in academic papers. This led us to update the architecture and protocol,

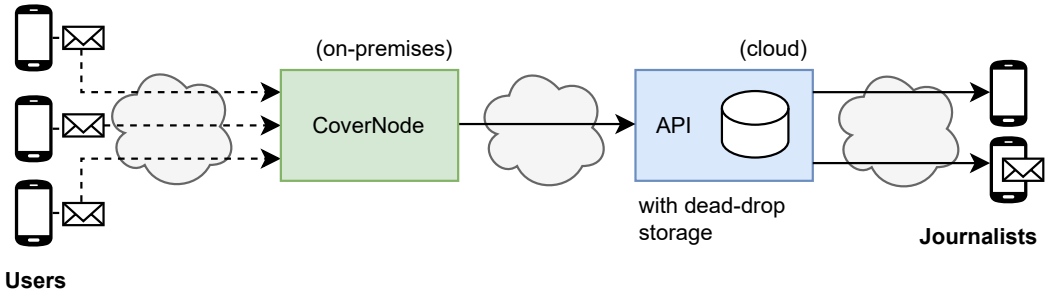

Figure 6.1: Overview of the CoverDrop system for messages from the users to journalists. The users' news reader apps send cover and real messages to the CoverNode which then mixes them and publishes signed dead-drops to the API. The journalist apps download the dead-drops and find messages addressed to them (if any). Own graphic that is also included in the white paper.

both of which are described in a not yet published white paper that we are writing together with the team at the news organisation. In this chapter, we discuss three important technical improvements that were made to the original design. These include added support for forward security (Section 6.2), more practicable scheduling of cover traffic (Section 6.3), and the efficient and secure PrivateSendingQueue (Section 6.4).

This chapter is based on the original paper "CoverDrop: Blowing the Whistle Through A News App" [2] and a white paper that has not been published yet. Both papers have been authored by many people and as such ideas in this chapter originate from collaboration. However, I can claim the following contributions: I reviewed and analysed available approaches for forward security; I proposed and implemented the new scheduling strategy; and I led the design, implementation, and analysis of the PrivateSendingQueue. All text in this chapter has been written independently which allows me to cover these topics in greater detail. The figures in this chapter contain specific declarations.

6.1 The CoverDrop system

We first give a high-level overview of the CoverDrop architecture and its components (Section 6.1.1) and introduce the underlying threat model (Section 6.1.2). Then, we summarise the requirements gathering process (Section 6.1.3).

6.1.1 Overview

In the CoverDrop architecture (Figure 6.1) all app users send cover traffic to a CoverNode operated by the news organisation such that sources which send actual messages can not be distinguished by a network adversary. They are effectively hidden within the very large user base. The CoverNode works as a mix node that filters out cover messages and releases real messages (padded with additional cover messages) in signed, fixed-sized outputs that we call dead-drops. These dead-drops are published by a web service and are publicly accessible. Journalists will download dead-drops and locally search for messages that are encrypted under their public key. Replies from the journalists travel through the system in reverse direction following the same principles and are published in user-facing dead-drops.

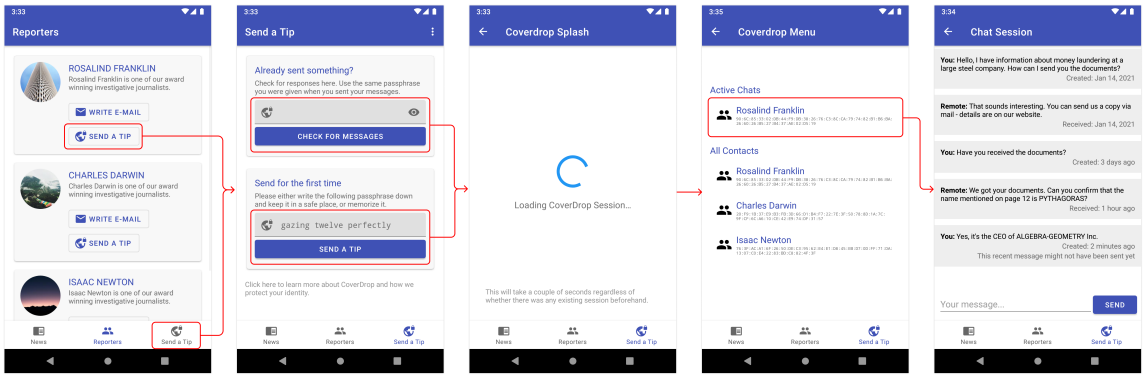

Figure 6.2: These screenshots show how a source would use the CoverDrop module within a news reader app. (i) They first navigate to the CoverDrop module following links in other content or the main menu. (ii) Since the storage uses plausibly-deniable encryption, the login screen always shows both an option to continue a session with an known passphrase and an option to create a new session under a new passphrase. (iii) After unlocking the encrypted storage, CoverDrop tries to decrypt messages within the cached dead-drops to find new messages addressed to the user. (iv) In the inbox the user selects one of their active chats. (v) The chat view shows the conversation history and allows to send new messages. Own graphic that is also included in the original CoverDrop paper. The fourth screenshot was adapted by hiding an UI element from the old sending strategy.

CoverDrop app module (all users). On every start of the news reader app the integrated CoverDrop module will download the public key hierarchy and all newly published user-facing dead-drops. It then verifies the signature chains of the hierarchy (Section 6.3), as well as the signatures of the downloaded dead-drops and stores them on disk. When the app is started for the very first time, a plausibly-deniable encrypted storage, e.g. HiddenSloth (Section 5.3), is initialised with a random passphrase. In addition, all users execute the background sending strategy (Section 6.3) every time the app is closed.

CoverDrop app module (for sources). When a user creates a new CoverDrop session, the CoverDrop module generates a randomly-chosen passphrase that the user has to memorise. As we discussed in the Sloth chapter, this is to prevent weak user-chosen passphrases. As the rate limiting by the SE provides strict time guarantees against an adversary, we can keep the passphrases short and memorable. Then, the plausibly-deniable encrypted storage is re-initialised with this new passphrase. During initialisation, the system stores a fresh user messaging key pair (Section 6.3) and a secret s_{PSQ} for the PrivateSendingQueue (Section 6.4) in the encrypted storage. Figure 6.2 shows how a source would use CoverDrop within the news reader app.

Every subsequent time the user unlocks their session using the passphrase, the CoverDrop module first tries to decrypt all messages from the cached dead-drops using the stored user messaging key pair and adds them to the internal state. When the user sends a new message or replies to a conversation, the outgoing message payload consists of its public key (so that the journalist can use it to encrypt their replies) and the message text that is first compressed and then padded to a fixed

length. This payload is then encrypted[1] in two layers: first under the journalist's public key and then under the public key of the CoverNode. The final cipher text is then added to the sending queue as discussed in Section 6.4.

CoverNode. The CoverNode runs on a dedicated on-premises machine and reads the incoming encrypted user messages from a message stream. Internally, the CoverNode works as a threshold mixer that consumes the incoming messages, decrypts their outer-most layer, filters out cover messages (signalled by a flag in the encrypted payload), and adds them to the output buffer. Whenever the CoverNode has read t_{in} messages, the output buffer is filled up with cover messages to a total size of t_{out} messages, signed by the identity key of the CoverNode, and published as a dead-drop via the web services. If there are more than t_{out} real messages, the excessive ones are held back for future rounds. As an additional defence in depth, the key management and mixing operations can be performed inside an SGX enclave to further harden the system against physical attacks.

Web services. In addition to the on-premises CoverNode, we run web services on third-party infrastructure. Importantly, we do not impose any confidentiality requirements for those services as all data processed by these services is end-to-end encrypted. However, like any other networking infrastructure, these services can affect availability. The central API service provides endpoints for downloading the public key hierarchy and published dead-drops. These are cached by the regular CDN of the news organisation which provides low-latency access for clients and makes it hard to filter CoverDrop traffic without also disrupting general news delivery. The CDN also operates an endpoint that takes the messages sent by the users and journalists and adds them to the dedicated messages stream from which they are later read by the CoverNode.

6.1.2 Threat model

For CoverDrop, we assume a strong adversary that can observe and record all network communication. This also covers an adaptive local adversary that wiretaps individual suspects. Note that this aligns with the typical adversary model for mix networks (Section 2.1). Recording all data is practical in our case as the estimated total traffic of unique encrypted messages only results in a few terabytes per day and is routed through a single message queue on third-party infrastructure.

We further assume that the adversary can demand access to the phone of a suspect at any time. This might happen at a border crossing or through the security staff at a workplace. As such the adversary can capture a single snapshot of the device and force the user to unlock the device which gives the adversary full access to all data stored on disk. However, we assume that the adversary cannot do so covertly and that the user will perform a full device reset after access by the adversary. This effectively yields a single-snapshot adversary model as in our threat model for HiddenSloth (Section 5.3). Similarly, we assume that the SE remains secure.

The adversary wins if they can gather evidence that a certain user is likely to have used the CoverDrop app module for sending messages to journalists. Therefore, both network communication

[1] A suitable approach is to derive a shared secret s using ECDH (using an ephemeral X25519 key pair and the recipient's public key) and then using s to encrypt the payload with an authenticated encryption scheme like XSalsa20-Poly1305. The outgoing bytes will comprise the public ephemeral key and the ciphertext including the MAC. The described construction matches LibSodium's SealedBox primitive.

and data stored outside the SE must provide plausible deniability. To counter censorship of the CoverDrop service, we require that restricting functionality of CoverDrop should be difficult without also denying access to other functionality and content of the app.

6.1.3 Requirements gathering

The CoverDrop project is not the result of academic contemplation, but originates from careful requirements gathering with investigative journalists and information security staff. This portion of the project includes a survey of existing contact options and two workshops. This part of the project was led by Diana. I briefly summarise the results in this section to motivate the trade-offs chosen for the implemented CoverDrop system. More details on the survey and the workshops can be found in the original CoverDrop paper [2] and Diana's PhD dissertation [141].

The survey reviewed the homepages of 24 news organisation of various countries based on their estimated popularity. We found that only half of the included news organisations offer encrypted communication channels such as E2EE messaging apps or dedicated solutions like SecureDrop [52]. SecureDrop is an anonymous communication and document sharing platform that is operated by the news organisation and allows access via a Tor hidden service. However, these solutions are not always easy to find and often require users to navigate through many links.

Based on these findings, 20 attendees were invited to a first workshop where the authors presented the survey results and shared ideas for a system that would later become CoverDrop. As it was important to both allow the attendees to provide additional information and ensure that all topics are covered, the discussion was guided in a semi-structured setting. This workshop yielded important insights:

- Cultivation of sources can be a long process during which the risk might change. For instance, the source might first reach out with only general remarks and then later share sensitive information. Hence communication should start in the most secure manner, as downgrading to other channels is possible, but retroactively removing previous metadata footprints is not.
- Current systems were described as high-latency and slow. This is for instance the case with SecureDrop which requires an operator to transport messages with an USB stick to an air-gapped computer. Such long delays can make it difficult for journalists to build trust and rapport with a potential source.
- Many systems are difficult to understand and thus make it hard to use them correctly. The attendees mentioned that it is important to find a good trade-off between security and usability. For instance, the solution should be available and self-contained so that potential sources do not accidentally draw attention to themselves through Internet searches using potentially revealing keywords.

After the first workshop a news organisation invited the team for a follow-up meeting that included a presentation of CoverDrop and open discussion. This meeting eventually focused on technical requirements such as utilising the CDN network and the ability of the in-house developer team to maintain and update the CoverDrop modules that are included in the apps. Also the risk of supply-chain attacks through external libraries and the inclusion of advertisement frameworks was

mentioned. As such, this meeting motivated many of the technical decisions of the CoverDrop architecture that are presented in this chapter.

6.2 Forward security in a high-latency anonymous messaging system

Modern end-to-end encrypted messaging protocols typically offer forward security and post-compromise security for conversations between two parties. *Forward security* limits the ability of an adversary which has compromised the internal state of the user's device to learn information about previously exchanged messages and *post-compromise security* makes it harder for the adversary to maintain the compromised state [38]. The CoverDrop setting imposes the additional challenge that such protocols must work well in a non-interactive and high-latency environment, as messages are delayed by the CoverNode and users might only unlock their local CoverDrop storage occasionally.

6.2.1 Security of messaging protocols

For the discussion in this section we consider an adversary that at some point in time compromises the internal state, including all key material, of a user. This could be the result of a vulnerability in the implementation or because the adversary gains physical access to the device that stores the information. However, there are also more subtle attack vectors, e.g. insufficiently secure backups, that can provide another opportunity through which an adversary might capture such a snapshot.

In a protocol with forward security a compromise of the internal state of one of the clients should not allow the adversary to decrypt messages that were exchanged sufficiently long ago. Otherwise, an adversary that routinely records all encrypted messaging traffic could "retrospectively wire-tap" conversations once they capture the phone of one of the parties. Following the terminology used by Boyd and Gellert [27], the delay after which messages can no longer be decrypted by the new key material, can be *windowed*, i.e. after a certain amount of time has passed, or *triggered*, i.e. after a certain sequence of events. Forward security is important for CoverDrop, as the total size of all exchanged cryptographic messages per day is only a few terabytes which would allow an adversary to record and indefinitely store all traffic at reasonable cost. Since device capture is part of our threat model, we believe that attempts to retrospectively access messages that formed the initial contact between source and journalist are plausible.

A protocol with post-compromise security makes it difficult for an adversary to maintain a state of compromise that allows them to decrypt all subsequent messages indefinitely. Ideally, we want the adversary to lose all access once they miss a single message during which the confidentiality of the communication session "self heals". This property is particularly interesting for long-running conversations and situations where the initial key exchange was performed in a less secure environment. Post-compromise security is of lower priority for CoverDrop, as it is primarily concerned with the initial contact between source and journalist. Therefore, it is likely that many conversations switch to alternatively channels relatively quickly.

6.2.2 Ratchet-based protocols and puncturable encryption

The Double Ratchet protocol [106] was the first widely deployed implementation that provides both forward security and post-compromise security[2]. This protocol was first deployed in Signal—therefore sometimes dubbed "the Signal protocol"—and since then it has been integrated in other popular end-to-end encrypted messaging apps. In addition to confidentiality of the messages, the Double Ratchet protocol comes with features that are important for real-world implementations such as allowing out-of-order messages and encryption of header information.

In this protocol each user maintains two Double Ratchets per communication partner: one for sending messages and one for receiving messages. These are mirrored between the communication partners so that A's sending ratchet matches B's receiving ratchet and the other way around. For every incoming or outgoing message the respective ratchet performs a symmetric ratcheting step using a KDF operation. This makes it hard to recover the previous state of the ratchet and thus key material that was used to encrypt old messages. In addition, both parties perform DH ratcheting steps by continuously generating new DH key pairs and exchanging the public parts with the other party. This introduces new DH shared secrets into the ratchets which will end compromise by an adversary unless the adversary actively intercepts these messages.

The Double Ratchet algorithm therefore requires interactive exchange of messages to protect the session. As such it is less suited for asynchronous communication systems, such as CoverDrop, where messages might take multiple hours before they reach the recipient. In addition, in our context the internal state is kept encrypted which further delays progress of the protocol until the user unlocks their CoverDrop session again. In particular, this rules out interactive DH key exchange for the initial session key, as doing so could delay receiving the first message payload by multiple days.

An alternative avenue for achieving forward security for a protocol with long-term public keys is puncturable encryption (PE) [60]. In a PE scheme the receiving party prepares a key pair that consists of a public key pk that is published and a secret key sk that is kept private. At any point the receiving party can update (or puncture) their secret key sk such that it can no longer decrypt a certain set of ciphertexts. In this context, ciphertexts are typically associated with tags based on content or epochs and then sk is punctured for these tags. Importantly, this can be done locally, non-interactively, and without having to publish new key material afterwards. Green and Miers present a practical forward-secure protocol based on PE [60]. They highlight that for existing PE schemes either the initially generated secret key sk is very large (proportional in the number of all possible tags); or the secret key sk grows with each performed puncture operations. While the latter is generally preferable, it grows rather quickly by multiple group elements for each puncture [60, Table I]. This not only increases the required storage, but also the decryption time which is linear in the key size $|sk|$. Hence, Green and Miers combine PE with a Hierarchical Identity Based Encryption (HIBE) scheme based on the work by Canetti et al. [29]. Using HIBE they can derive fresh PE keys without any existing punctures for new epochs; at the same time the updated HIBE secret key can no longer derive keys for the same epoch again. By choosing sufficiently short interval durations, they avoid that the PE keys grow very large since they expect only a limited number of puncture operations per epoch.

Even relatively low numbers of around 100 punctures can lead to decryption times of around 2 seconds per message on mobile [60, §IX]. This is not practical for the current CoverDrop protocol

[2] In their paper this property is called "break-in recover".

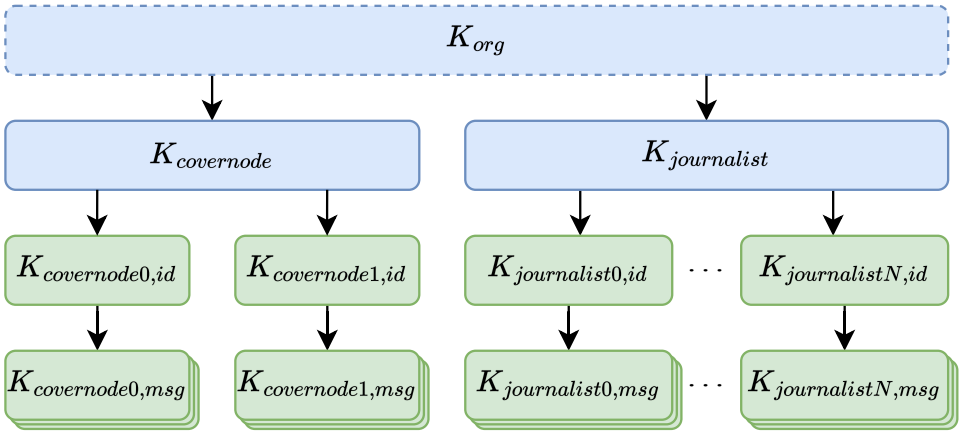

Figure 6.3: The news organisation key hierarchy based on the current organisation key K_{org}. The blue elements are stored on compartmentalised services by the news organisation (and K_{org} is stored offline as indicated by the dashed lines). The green elements are keys that are stored by the individual CoverNodes and journalists on their devices. The arrows point from the signing key to the signed key. Own graphic that is also included in the white paper.

where journalists try decrypting many messages from the dead-drops to find the ones addressed to them. Further, this opens the potential by adversaries to perform a denial of service (DoS) attack where they send n messages to a journalist leading to $\mathcal{O}(n^2)$ decryption effort[3]. Finally, we are not aware of a production-ready implementation that is available on all platforms that we need to support.

6.2.3 Key rotation and management

For CoverDrop we adopted a multi-level key rotation scheme that is based on the key hierarchy shown in Figure 6.3. We use it to allow the keys in the lower levels in the hierarchy to rotate quickly. Keys are signed by their parent keys, e.g. $K_{journalist,msg}$ is signed by $K_{journalist,id}$, before they are published to the API. The signature for each published key spans both the public key itself and its expiry date ensuring that the their validity can be verified by the client as well. The API serves a complete snapshot of all valid keys, excluding expired ones, to the clients.

The key hierarchy is created with the organisation key K_{org} as its root of trust. This key is stored offline and only directly signs the provision keys $K_{covernode}$ and $K_{journalist}$ in rare, manual key ceremonies. As an additional security measure, the offline key can be split using Shamir's Secret Sharing [121] and distributed amongst staff members. Key ceremonies are used in practice to protect sensitive key material, e.g. the Domain Name System Security Extensions (DNSSEC) Root Key Signing Key [72]. The organisation key K_{org} is directly embedded in the app so that for the clients its security aligns with the security of the app distribution. As such, introducing a new organisation key requires updating and publishing a new version of the app. The news organisation can optionally print the public organisation key in their newspaper to allow manual out-of-band verification by the user.

[3]The quadratic decryption effort stems from the observation that for the $(i+1)$-th message there have been already i punctures resulting in key size (and decryption effort for this message) proportional to i. Therefore, $\sum_{i=1}^{n} i-1 = \mathcal{O}(n^2)$.

The provision keys $K_{covernode}$ and $K_{journalist}$ are typically valid for multiple months and are used to sign the identity keys for the CoverNodes and journalists, respectively. Both keys are accessed through dedicated identity services that allow authorised staff members to create new initial identity keys for CoverNodes and journalists. Once these initial identity keys are in use by journalists (or the CoverNodes), they can self-issue themselves new identity keys. For this the journalist creates a new identity key pair $kp' = (pk', sk')$ and signs a challenge $timestamp \parallel pk'$ with their current identity key sk. The signed challenge is then sent to the identity service which verifies it and subsequently publishes the included new identity public key pk' together with a signature under the provisioning key $K_{journalist}$ to the API. This protocol works analogously for the CoverNode keys under $K_{covernode}$. Having dedicated provisioning keys $K_{covernode}$ and $K_{journalist}$ allows the sensitive K_{org} to remain offline and the identity services can be strongly compartmentalised and access restricted to CoverNodes and journalists on the organisation's internal network.

Identity keys expire after around a month which makes them useful for long-lived signatures, e.g. the CoverNode signing the dead-drops it publishes. However, they are too coarse for effective forward security. Therefore, journalists generate fresh messaging keys every day and publish them with a signature from their identity key to the API. Each messaging key has a validity period of 14 days, such that journalists can be offline for a few days and still decrypt messages that were sent to them during this time when they come back.

As a result, the lifetime of the messaging keys overlaps and clients should always use the one with the longest remaining lifetime. Similarly, there may exist multiple valid identity, provision, and organisation keys for the same subjects at the same time. This is because journalists and CoverNodes should publish new keys before the existing ones expire to ensure that there is always at least one valid key available to the clients. However, this also means that the clients of the journalists and users need to carefully traverse all resulting trees and signature chains to collect the total set of valid messaging keys when trying to decrypt messages from the published dead-drops.

Our approach fits the "windowed" type for forward security in the classification by Boyd [27]. It is driven by the need for a non-interactive protocol due to the high-latency and unavailability of key material on the clients when in an locked state. The window of opportunity for an adversary is two weeks which seems acceptable and can be easily changed by updating the life-time for messaging keys.

A service similar to Certificate Transparency [85] can be deployed to monitor for unexpected keys that might indicate that an adversary tries to perform a key equivocation attack or tries to register a fake journalist. In addition, such a monitoring service can warn the engineering team if expected key rotations are not happening and part of the tree is at risk of expiring.

6.3 User-friendly message scheduling on Android and iOS

When a user writes and sends a new message in the CoverDrop module, it is first encrypted and then added to a queue for sending it to the CoverNode at a later time. Sending messages independently from when they are created is important to ensure that we do not leak when a potential whistleblower uses CoverDrop. In particular, the timing of cover messages and real messages should be indistinguishable for an outside observer.

On mobile devices the operating system mediates access to resources more comprehensively than on desktop computers. This is necessary as otherwise a single app could perform excessive background operations and quickly deplete the phone's battery. Hence, scheduling of background tasks happens exclusively through the APIs provided by the platform. This allows the OS to limit execution times for each app and schedule, e.g. multiple background tasks to run in parallel which allows them to share an already established radio connection.

As we mentioned in Section 3.1.3, Android's background scheduling is based around the concepts of Doze and App-Standby [54]. *Doze* reduces the number of times the phone wakes up for regular background tasks across all apps, while *App-Standby* identifies infrequently used apps and drastically limits their allowance for background tasks up to and including never running them at all. iOS has similar restrictions imposed by its Background Task API [7].

The original CoverDrop paper proposes a simple schedule where the background sending task is executed every hour throughout the day. Each execution removes the front-most message from the queue and sends it to the message queue endpoint that runs under the news organisations domain. This ensures that the sending operation is fully decoupled from the active usage of CoverDrop and thus provides strong plausible deniability of the sent traffic.

However, in practical experiments we found that both platforms make it difficult to reliably and predictably schedule periodic background events. Our observations are compatible with the findings of the Don't Kill My App project [97] which found that the majority of Android smartphones add their own heuristics on top of the background logic that is already part of the base AOSP image. As a result, many third-party applications, such as alarm clocks, do not work correctly on many phones or require manual intervention by the user to disable the default energy and performance "optimisations". For our use-case this means messages might not be sent at all. This effect can be biased towards users who open the app very infrequently and are not interested in performing manual steps to make background tasks run more reliably.

Based on the practical restrictions, we adopted a combined sending strategy that provides reliable message delivery and reasonable independence between CoverDrop sessions and the background send events. The integration of our strategy executes when the user closes or otherwise leaves the app. This integration code schedules one execution of the background task after a randomly chosen delay d and sets a flag to mark execution of the background task as pending. When the background task runs, it first sends the configured number of messages from the queue and then clears the flag.

If later the app is started and the flag is still set, this indicates the background task was not (yet) executed. As a fall-back, the sending task is executed while the app loads. This fall-back mechanism ensures that messages are eventually delivered even when background jobs are completely unreliable. Observing that messages are sent on app start does not give any meaningful information about CoverDrop usage to an observer, as the timing and conditions for execution are independent of whether real message were scheduled. In addition, we use a local rate limit, e.g. not more than 6 messages every 3 hours, to limit the traffic consumed by CoverDrop even if a user opens and closes the app frequently. If a send event would exceed this limit, it is ignored and the message remains in the queue ready to be sent at the next available opportunity.

We sample the delay d from a random distribution $Exp(\Lambda)$. Reasonable choices for Λ are in the order of $1\,\text{hour}^{-1}$, i.e. the expected mean delay is 1 hour. This random delay is important to obfuscate

network traffic gaps between reading the last news article t_{read} and sending the queued messages t_{send} which are both events observable by the adversary. If we would not choose a random delay, i.e. d is constant, then for regular users the duration $t_{send} - t_{read}$ would be shorter than for a user who spends some time in CoverDrop before closing the app.

Scheduling the sending event after app exits minimises the duration that real messages spend in the queue, as they are typically sent soon after they were enqueued. As further optimisation, the app might check in the background task if the phone is, e.g. connected via WiFi, and send more messages in this case.

6.4 An efficient private sending queue

In the original CoverDrop paper, encrypted real messages, which are scheduled to be sent, are queued as ciphertexts on disk. They remain on disk until the background process picks them up and sends them to the CoverNode. During this time the user is at risk in the case where their phone is captured by an adversary, because the existence of these ciphertexts is proof that they indeed used CoverDrop recently to send real messages. While the time window for such attacks might be short, it is nevertheless a real risk as we can expect that, e.g. their employer can access phones of their employees during the day with short notice. We therefore design an improved PrivateSendingQueue feature that can hide the number of real message that are currently waiting to be sent.

6.4.1 Overview and threat model

For the sending queue we assume an adversary that can capture the phone right after a source used CoverDrop—or interrupts them while using it, forcing them to quickly close the app. The adversary wants to determine whether a person used CoverDrop to reach out to journalists, i.e. whether there is at least one encrypted real message waiting to be sent. As such they can capture an initial snapshot S_0 of the phone right after the user left the app. With full control of the device they can also trigger the background job multiple times, giving them the opportunity to capture snapshots $S_1, S_2, \dots$ right after the first, second, etc. execution of the background job.

In the protocol from the original paper, the adversary wins by simply checking for a non-empty queue in any of the $S_0, S_1, \dots$ snapshots. An improved version could, for instance, always fill up the queue with a random number of additional cover messages—both when adding real messages in the app and when dequeueing message in the background task. However, this method leads to longer delays when sending real messages as they might get queued behind cover messages. Also, this approach still leaks information. When enqueueing real messages from within the CoverDrop session, the queue most likely will not be empty. Therefore, after adding the real messages (even when not adding any additional cover messages) the queue in S_0 (and subsequent $S_1, \dots$) will likely be on average slightly longer than on a phone where CoverDrop was not used. While this no longer gives the adversary a definite proof, the adversary might consider a person with a larger queue more likely to be a source than other users.

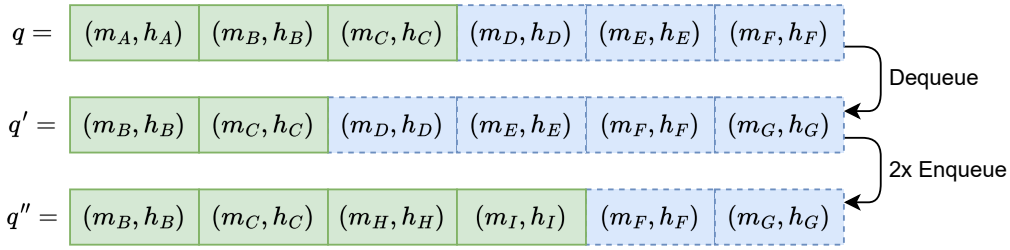

Figure 6.4: Illustration of a PrivateSendingQueue of size $|q| = 6$ and execution of three operations. Green indicates real messages and blue (dashed) indicates cover messages. The first instance q comprises three real messages and three cover messages, i.e. $f = q.\text{FillLevel} = 3$. After dequeueing the front-most item, e.g. by the background task, the resulting queue q' has its elements shifted and a new cover item (m_G, h_G) is added. Enqueueing two items, e.g. from within a CoverDrop session, results in queue q'' where the two front-most cover items are replaced by the new messages m_H and m_I. Own graphic not included in either paper.

6.4.2 Construction

We propose the PrivateSendingQueue (PSQ) as a solution that hides the number of enqueued real messages from the adversary and ensures that real messages are always sent before cover messages. The PSQ is persisted as a queue of tuples $q = [(m_1, h_1), (m_2, h_2), \ldots, (m_{|q|}, h_{|q|})]$ consisting of the encrypted message ciphertexts m_i and their hints $h_i \in \{0,1\}^b$ where b is a freely chosen but fixed security parameter. Some operations require a secret key s that should be stored inside the encrypted storage so that it is not available to the adversary. The PSQ also requires a supplied function NewCoverMessage that generates an encrypted cover message that is indistinguishable from an encrypted real message. Creating such a message outside an unlocked CoverDrop session is possible as all required key material is available from the cached key hierarchy. Figure 6.4 illustrates a sample PSQ as it undergoes dequeue and enqueue operations.

Our PSQ always maintains three invariants: The *size invariant* requires that its length is constant $|q| = l_q$. The *order invariant* requires that for all real messages m_i there is no cover message m_j with $j < i$, i.e. real messages are always at the front of the queue. The *hint invariant* requires that for all real messages m_i, the hint h_i is equal to the b-bit-long output of a secure keyed hash function, e.g. HMAC with SHA-256, under the secret key s; otherwise, the hint must be a bit string of length b chosen at random.

The PSQ.Init method creates a new list by assigning each message a new cover message $m_i \xleftarrow{\$}$ NewCoverMessage() and each hint a randomly chosen bit sequence $h_i \xleftarrow{\$} \{0,1\}^b$ for all $i \in [1, |q|]$. The initial queue satisfies all invariants.

The PSQ.Dequeue operation removes the front most tuple (m_1, h_1) which causes all following items to advance $(m_i, t_i) \leftarrow (m_{i+1}, t_{i+1})$. It then adds a new tuple consisting of a cover message $m_{|q|} \xleftarrow{\$}$ NewCoverMessage() and a randomly chosen hint $h_{|q|} \xleftarrow{\$} \{0,1\}^b$. We assume inductively that the PSQ fulfilled both invariants before the PSQ.Dequeue operation. As one element is removed and one is added, the PSQ still maintains the size invariant. And because the new element is a cover message with a randomly chosen hint and added as the last item, the hint and order invariants are maintained as well.

The PSQ.FillLevel operation returns the number r of real messages given the secret key s_{PSQ}. For this the operation iterates through the tuples (m_i, h_i) of the list starting from the beginning of the queue, i.e. $i = 1, 2, \ldots, |q|$. While $h_i = \text{HMAC}(s, m_i)$, it increments i by one and continues the loop if $i \leq |q|$; otherwise the loop terminates. The operation then returns $r = i - 1$. All invariants of the list are maintained, as no changes are being made.

The PSQ.Enqueue operations adds a real message m to the queue given the secret key s_{PSQ}. It first determines the fill level $f = \text{Q.FillLevel}(s)$. If $f = |q|$, the queue is full with real messages and the operation aborts with an error; leaving all invariants intact. Otherwise, it computes $h \leftarrow \text{HMAC}(s_{\text{PSQ}}, m)$ and sets the front-most cover message tuple to the new one for the real message: $(m_f, h_f) \leftarrow (m, h)$. Because we replace a cover message with a real message and its correct hint, the size and hint invariants are maintained. Inserting the message right at the fill level position, i.e. after the right-most real message, ensures that there are no cover messages before it, thus satisfying the order invariant.

The hints of the PSQ also allow to indicate to the user which messages have already been sent. For this the persisted chat history also stores for each message the respective hint h that was computed when inserting it into the PSQ. When a user later re-opens the chat view, all messages whose hints are still in the PSQ can be marked as *pending* while the others are marked as *sent*.

There is the possibility that a randomly chosen hint h for a cover message m actually matches the $\text{HMAC}(s_{\text{PSQ}}, m)$ value that would be calculated for a real message. For practical concerns, any security parameter $b > 80$ renders the chance of this happening negligible. In case it happens nevertheless, this collision does not affect security, but only affects efficiency as the PSQ.FillLevel operation would return a too high value which then can cause a real message to be enqueued after a cover message. Such a queue would violate the order invariant until the bogus (m, h) is removed by subsequent PSQ.Dequeue operations.

6.4.3 Security analysis

We use common assumptions about indistinguishability and CPA security as well as the maintained PSQ invariants. We want to show that an adversary cannot determine the number of real messages in the queue better than random guessing.

We assume that without access to the secret key s_{PSQ} the output of HMAC is indistinguishable from random bitstrings of the same size. This is generally believed to be true for commonly used instantiation with, e.g. using SHA-256, as Bellare [22] showed that a HMAC is a PRF given the underlying compression function is a PRF. Therefore, access to the hints $h_{\ldots}$ without s_{PSQ} does not give the adversary any advantage.

The encrypted messages comprise an ephemeral public key eph_{pub} that is used for a ECDH key agreement deriving the shared secret s; the ciphertext encrypted under s using a CPA-secure authenticated encryption scheme; and the randomly chosen nonce. Both real and cover messages are generated using the same procedures and therefore both will contain a randomly generated eph_{pub} and a randomly generated nonce. Under the generalised decisional Diffie-Hellman assumption the adversary cannot derive the s without access to either the sender's eph_{priv} or the recipients private key. By definition CPA-secure encryption schemes do not leak information about the encrypted content. Therefore, access to the messages $m_{\ldots}$ does not give the adversary any advantage.

We convinced ourselves when describing the operations in Section 6.4.2 that all three PSQ invariants are maintained at all times. The PSQ size invariant ensures that the queue, once initialised at the first app start, always has the same number of elements, thus there are no changes in size that could leak information. Therefore, we conclude given these arguments that the PrivateSendingQueue maintains confidentiality of the number of included real messages against the described single-snapshot adversary.

6.5 Summary

In this chapter we discussed CoverDrop as a practical system for whistleblowing that is integrated into a news app with strong plausible deniability against both network and local adversaries. Starting from an existing prototype, we presented three improvements to the security and efficiency of the mobile app. First, we showed that for high-latency messaging between devices, which are online only occasionally, key rotation is a practical approach for achieving forward security. In particular, solutions that rely on interactive key exchange are less suitable for highly asynchronous communication. We then revisited how cover traffic is scheduled for the news reader app and adapt it to the restrictions imposed by mobile platforms. As the third improvement, we designed the PrivateSendingQueue that hides how many real messages are currently waiting to be sent while also ensuring that real messages are always sent before cover messages.

Chapter 7

Conclusion and future work

In this dissertation we explored the technical frontier of metadata private systems on mobile devices. We first measured the impact of anonymity networks on the battery life and concluded that low-latency protocols with cover traffic appear infeasible. In this chapter we reflect on this result by looking at the protocol layer integration and suggest venues for improvement. In Chapter 4 we introduced the Rollercoaster protocol which mitigates some of the technical limitations of Loopix to provide more efficient multicast support. We will discuss here how Rollercoaster can be combined with other building blocks to create decentralised applications with strong metadata privacy. Both Sloth and CoverDrop worked around restrictive APIs on smartphones through novel designs for strong key stretching and background cover traffic. However, we believe that even better solutions are possible with the right APIs. Based on these observations we argue that tighter integration between layers of the communication stack and more flexible use of the Secure Element can unlock more usable and more efficient systems.

On the energy consumption of anonymous communication protocols on mobile devices. We hope that our easy-to-use and replicable measurement setup enables more researchers to investigate the energy consumption of anonymity network implementations on smartphones which arguably are the most important personal computing platform today. While some existing implementations, such as Tor, are feasible today, we learned that running low-latency anonymity networks with strong metadata privacy at reasonable costs remains an unsolved problem. One might be tempted to extend the Anonymity Trilemma [46] with energy consumption: "Strong anonymity, low latency, efficient bandwidth usage, and energy efficiency—choose two." Overall, it becomes clear that choosing sensible and well-argued parameters to achieve application-specific trade-offs remains an important aspect of all work in this area.

Our measurements point to a large potential for improvement by better integrating the application logic with the underlying radio communication and hardware. However, approaching this solely from the application perspective is difficult as the internal state machine of the radio modem cannot be easily inspected and operates as a black box. Instead, there is need for an API provided by the platform. Such an API could expose some view of the inner state and configuration of the radio modem to allow the application to make better decisions. For example, it could indicate the available

theoretical throughput and tail-latency parameters. Alternatively, the API could allow applications to mark their communication requirements, e.g. low-latency, regular, or high-throughput, and then adopt the radio configuration accordingly. For instance, for a low-latency small-payload transmission, the data might be able to be transported via the signalling channel rather than via a data connection that needs to be established first.

The lack of such an API, as well as the restrictive nature of the background tasks, hint at the difficult position these platforms find themselves in; and where they have to mediate between users, developers, and limitations of the available hardware. For instance, app developers want to to build innovative applications with as few restrictions as possible, but at the same times users demand predictable battery life and that the operating system protects them from misbehaving apps. The historical development of Android illustrates how the platform first added features to help identify power hungry apps and then subsequently imposed more explicit restrictions. This indicates that relying on self-regulating behaviour from many thousands of app developers did not work and centralised enforcement by the platform might be necessary after all. Therefore, providing app developers with more control over the radio hardware appears unlikely in this context.

Alternatively, we can optimise efficiency by more tightly integrating the end-user application and the underlying anonymity network protocol—both of which run above the platform layer and therefore are under our control. For example, an anonymous messaging app might want to send an urgent message right away outside the (e.g. Poisson-based) sending schedule. This would effectively trade "some anonymity" to allow single low-latency transmissions while still choosing conservative parameters that benefit overall battery life. However, to our knowledge, we lack a good understanding of how these out-of-order messages affect anonymity in different adversary settings. It is also unclear how such an approach is best implemented to ensure long-term invariants, e.g. compensating for a shortened delay by having a longer delay in the future such that the average sending rate remains the same. Another approach is, of course, to adapt the application and protocol layers to work more efficiently within the present limitations as we showed with Rollercoaster.

One example of a very tight integration between use-case and anonymity network is CoverDrop. For the CoverDrop app the energy efficiency is particularly important since all app users help contribute to the anonymity set voluntarily without immediate benefit. The results from our measurements showed that we can effectively ignore the cryptographic costs and should instead focus our time on choosing efficient communication patterns. As such, the results of Chapter 3 can inform better allocation of engineering resources.

On anonymous, decentralised group-based applications on smartphones. The field of decentralised group communication with strong metadata privacy remains an interesting research area. In particular, Rollercoaster by itself does not immediately yield useful and secure applications, but it rather is a building block that can incorporated into higher-level protocols and applications.

For example, for practical group communication all members will also need to agree on keys and update these regularly. Key updates are important to provide forward security [27] and post-compromise security [38] in order to reduce the risk when device state gets compromised. Existing standards like MLS [19] require a central server that establish a total order of messages. However, such a central server is able to observe group communication metadata, which violates the metadata privacy

that we try to achieve. Weidner et al.[1] proposed the Decentralized Continuous Group Key Agreement (DCGKA) protocol [146] that works without a central server and gracefully handles asynchronous communication. DCGKA requires for efficient operation that the underlying transport layer provides a broadcast primitive—which Rollercoaster can provide. In fact, this was one of the considerations while designing Rollercoaster. Together a stack consisting of Loopix, Rollercoaster, and DCGKA can provide anonymous E2EE group communication.

However, this instantiation leaves out the important aspects of real-world group membership management. In particular, as new group members are added and existing ones are removed, we might want to ensure that these operations can only be performed by a trusted subset of group members. For this research question, I supervised the thesis "Private Group Management (PGM) for Mix Networks" [119] that proposes an extension on top of Rollercoaster which allows dedicated group administrators to change group membership while simultaneously updating the Rollercoaster distribution graph.

We believe that these building blocks—Rollercoaster, DCGKA, and PGM—allow us to build practical applications on mix-based anonymity networks with very strong privacy guarantees. One interesting direction is local-first software [80] where application state changes, e.g. edits to a text document, are applied and stored locally on each participants device and then synchronised between collaborators. This synchronisation can be performed using Conflict-free Replicated Data Types (CRDTs) [122] that allow to merge concurrent changes without the need for a central server. Therefore, CRDT-based applications are compatible with running on top of our metadata-private stack using Loopix, Rollercoaster, DCGKA, and PGM. For instance, we can build a metadata-private collaborative text editor for journalists who work together on a sensitive news story. Because CRDTs handle concurrent changes well, they improve the application's tolerance for high-latency communication which in turn allows us to choose power efficient parameters for the underlying anonymity network.

On securely storing data for anonymous decentralised applications on on smartphones. In decentralised settings, e.g. the suggested CRDT-based applications above, we need to securely store application and protocol state locally. Even if we use encrypted backups, at least some key material and identifier information remains on the device; and because mobile devices are always at risk of being stolen or otherwise accessed by others, keeping the local data secure is essential.

For applications where strong metadata privacy is important, we typically assume a strong dedicated adversary which can lead to situations where a user might be forced to unlock their device. In these situations, brute-force resistant deniable encryption, such as HiddenSloth, is an important feature that allows users to deflect suspicions and remain safe. While our strict time guarantees in the Sloth schemes allow for short passphrases, they do not match the comfort of, e.g. biometric authentication. If we would be able to run more complex programs instead of simple operations on the SE, we believe that the trade-off between usability and plausibly-deniable encryption can be drastically improved.

[1] I am one of the co-authors.

On improving the utility of SEs of smartphones. Third-party access to the SE is restricted for good reasons. It minimises the overall attack surface and reduces the risk of side-channel attacks from code running inside the SE. After all, it is the limited complexity and strong isolation that ensure that SEs remain secure in adversarial settings. However, with Sloth we also saw that very narrow APIs can hinder development of new solutions that provide additional security properties.

Under these considerations it seem unlikely and risky to simply run user-supplied binaries directly on the SE. At the same time, adding many specialised functions for each newly identified use case quickly leads to a vast API surface which is hard to audit and which will never be truly cover *all* possible application needs. Instead, it might be interesting to allow developers to submit simple programs or descriptions thereof to the SE. These could be modelled as computation graphs consisting of cryptographic primitives and simple control flow instructions, e.g. if-statements. These provide an expressive yet limited interface for application developers, allowing them to implement algorithms such as attempt counters or custom remote attestation. Using graphs also allow for performing simple static analysis. By requiring acyclic graphs we can guarantee termination, i.e. rule out loops. And by measuring the length of all paths from the entry node on the instruction level, we can verify that an algorithm has constant runtime and is less vulnerable to side-channel timing attacks.

On the CoverDrop project. The CoverDrop project is an encouraging example of research that emerged from a user-centered design process instead of being a "solution looking for a problem". This requirements gathering process informed the project direction and contributed to the fact that it is now being implemented. The implementation process in turn yielded many interesting research questions that can only arise when considering the technical limitations of a real-world deployment.

The three examples discussed in Chapter 6 are representative of most challenges that we encountered and which stem from the limitations of the underlying systems and scope of the undertaking. In particular, we found that we are not limited by the features and guarantees offered by widely-used cryptographic algorithms, but rather the main challenge is the management of key material in a multi-stakeholder environment. In that context the often repeated saying that cryptography just replaces a data privacy problem with a key management problem rings true. We think that fail-over behaviour, on-premises deployment, and key ceremonies find too little attention in current research.

Concluding thoughts. This dissertation discussed aspects of metadata private communication on mobile devices. It is encouraging that popular anonymity networks, such as Tor, appear feasible and ready for wider adoption. However, strong metadata privacy for low-latency applications remains out-of-reach. Achieving truly private and feasible solutions will involve a focus on handling asynchronous communication and tight collaboration across the stack from radio chip to user interface.

Many challenges that we addressed in this thesis were discovered when we hit technical limitations that prevented naïve approaches from succeeding. These include the limited energy supply of smartphones, the rate-limit of Loopix' outbound queue, the narrow APIs of SEs, and the restrictions for background tasks on mobile operating systems. Hitting and exploring these limitations proved to be a powerful tool for discovering impactful research areas and informs constraints for more theoretical work upstream. At the same time, some of these limitations are human made and as such they are malleable through communication and collaboration.

Bibliography

[1] Mehdi Ahmed-Nacer, Claudia-Lavinia Ignat, Gérald Oster, Hyun-Gul Roh, and Pascal Urso. Evaluating CRDTs for real-time document editing. In *Proceedings of the 11th ACM Symposium on Document Engineering*, pages 103–112. ACM, September 2011.

[2] Mansoor Ahmed-Rengers, Diana A Vasile, Daniel Hugenroth, Alastair R Beresford, and Ross Anderson. Coverdrop: Blowing the whistle through a news app. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*, pages 47–67, 2022.

[3] Amazon. AWS Device Farm, 2023. Accessed September 2023. URL: `https://aws.amazon.com/device-farm`.

[4] Ross Anderson and Eli Biham. Two practical and provably secure block ciphers: BEAR and LION. In *Proceedings of the International Workshop on Fast Software Encryption*, pages 113–120. Springer, 1996.

[5] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with compressed queries and amortized query processing. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy (S&P '18)*, pages 962–979, 2018.

[6] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, pages 551–569, 2016.

[7] Apple Inc. Advances in app background execution, 2019. Accessed September 2023. URL: `https://developer.apple.com/videos/play/wwdc2019/707/`.

[8] Apple Inc. Apple platform security - Secure Enclave, 2021. Accessed September 2023. URL: `https://support.apple.com/en-gb/guide/security/sec59b0b31ff/web`.

[9] Apple Inc. App Store - iOS and iPadOS usage, 2022. Accessed September 2023. URL: `https://web.archive.org/web/20230130101543/https://developer.apple.com/support/app-store/`.

[10] Apple Inc. Apple security bounty, 2022. Accessed September 2023. URL: `https://security.apple.com/bounty/`.

[11] Apple Inc. Choosing background strategies for your app, 2022. Accessed September 2023. URL: `https://developer.apple.com/documentation/backgroundtasks/choosing_background_strategies_for_your_app`.

[12] Apple Inc. Apple developer documentation - SecureEnclave, 2023. Accessed September 2023. URL: `https://developer.apple.com/documentation/cryptokit/secureenclave`.

[13] Apple Inc. Apple developer documentation - SecureEnclave.P256, 2023. Accessed September 2023. URL: `https://developer.apple.com/documentation/cryptokit/secureenclave/p256`.

[14] Diego F Aranha, Pierre-Alain Fouque, Chen Qian, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. Binary elligator squared. In *Proceedings of the International Conference on Selected Areas in Cryptography*, pages 20–37. Springer, 2014.

[15] Luca Ardito, Giuseppe Procaccianti, Marco Torchiano, and Giuseppe Migliore. Profiling power consumption on mobile devices. *ENERGY*, pages 101–106, 2013.

[16] Arm Limited. *Arm Instruction Set Reference Guide*, 2018. Version 1.0 (100076_0100_00_en). URL: `https://documentation-service.arm.com/static/6245c734b059dc5ff9a8bdab`.

[17] Feng Bao, Robert H Deng, and Huafei Zhu. Variations of Diffie-Hellman problem. In *Proceedings of the 5th International Conference on Information and Communications Security (ICICS '03)*, pages 301–312. Springer, 2003.

[18] Ludovic Barman, Moshe Kol, David Lazar, Yossi Gilad, and Nickolai Zeldovich. Groove: Flexible metadata-private messaging. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)*, pages 735–750, 2022.

[19] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) protocol. RFC 9420, July 2023. URL: `https://www.rfc-editor.org/info/rfc9420`.

[20] Filipe Beato, Kimmo Halunen, and Bart Mennink. Improving the Sphinx mix network. In *Proceedings of the International Conference on Cryptology and Network Security (CANS '16)*, pages 681–691. Springer, 2016.

[21] Karoline Meta Beisel, Constanze von Bullion, Lara Fritzsche, and Nicola Meier. Handy-Jahre einer Kanzlerin. *Süddeutsche Zeitung Magazin*, 2021. Accessed September 2023. URL: `https://www.reporterpreis.de/upload/fritzsche-kanzlerin-6151d653c26ce.pdf`.

[22] Mihir Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In *Crypto*, volume 4117, pages 602–619. Springer, 2006.

[23] Daniel J Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (ACM CCS '13)*, pages 967–980, 2013.

[24] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: new generation of memory-hard functions for password hashing and other applications. In *Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P '16)*, pages 292–302. IEEE, 2016.

[25] Robert Bodle. The ethics of online anonymity or Zuckerberg vs. Moot. *ACM SIGCAS Computers and Society*, 43(1):22–35, 2013.

[26] Joseph Bonneau. Deep dive: EFF's new wordlists for random passphrases. *Electronic Frontier Foundation (EFF)*, 2016. Accessed September 2023. URL: `https://www.eff.org/deeplinks/2016/07/new-wordlists-random-passphrases`.

[27] Colin Boyd and Kai Gellert. A modern view on forward security. *The Computer Journal*, 64(4):639–652, 2021.

[28] Daniel R. L. Brown. SEC 1: Elliptic Curve Cryptography. Standard, Certicom Research, May 2009. Version 2.0. URL: `https://www.secg.org/sec1-v2.pdf`.

[29] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *Advances in Cryptology–EUROCRYPT*, pages 255–271. Springer, 2003.

[30] Aaron Carroll, Gernot Heiser, et al. An analysis of power consumption in a smartphone. In *Proceedings of the USENIX Annual Technical Conference*, volume 14, pages 21–21, 2010.

[31] Bing Chang, Fengwei Zhang, Bo Chen, Yingjiu Li, Wen-Tao Zhu, Yangguang Tian, Zhan Wang, and Albert Ching. Mobiceal: Towards secure and practical plausibly deniable encryption on mobile devices. In *Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '18)*, pages 454–465. IEEE, 2018.

[32] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[33] David Chaum, Debajyoti Das, Farid Javani, Aniket Kate, Anna Krasnova, Joeri De Ruiter, and Alan T Sherman. cmix: Mixing with minimal real-time asymmetric cryptographic operations. In *Proceedings of the International Conference on Applied Cryptography and Network Security*, pages 557–578. Springer, 2017.

[34] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[35] Chen Chen, Daniele E Asoni, David Barrera, George Danezis, and Adrain Perrig. HORNET: High-speed onion routing at the network layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (ACM CCS '15)*, pages 1441–1454, 2015.

[36] Raymond Cheng, William Scott, Elisaweta Masserova, Irene Zhang, Vipul Goyal, Thomas Anderson, Arvind Krishnamurthy, and Bryan Parno. Talek: Private group messaging with hidden access patterns. In *Proceedings of the Annual Computer Security Applications Conference*, pages 84–99, 2020.

[37] Nicolas Christin. Traveling the Silk Road: A measurement analysis of a large anonymous online marketplace. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 213–224, 2013.

[38] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. On post-compromise security. In *Proceedings of the 29th IEEE Computer Security Foundations Symposium (CSF '16)*, pages 164–178, 2016.

[39] Competition and Markets Authority. Investigation into Apple AppStore, 2023. Accessed August 2023. URL: `https://www.gov.uk/cma-cases/investigation-into-apple-appstore`.

[40] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy (S&P '15)*, pages 321–338, 2015.

[41] Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (ACM CCS '10)*, pages 340–350, 2010.

[42] George Danezis and Ross Anderson. The economics of resisting censorship. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (S&P '05)*, pages 45–50. IEEE, 2005.

[43] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy (S&P '03)*, pages 2–15, 2003.

[44] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *Proceedings of the 2009 IEEE Symposium on Security and Privacy (S&P '09)*, pages 269–282, 2009.

[45] George Danezis and Andrei Serjantov. Statistical disclosure or intersection attacks on anonymity systems. In *Proceedings of the International Workshop on Information Hiding*, pages 293–308. Springer, 2004.

[46] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. Anonymity trilemma: strong anonymity, low bandwidth overhead, low latency-choose two. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy (S&P '18)*, pages 108–126, 2018.

[47] Alex Davidson, Gonçalo Pestana, and Sofía Celi. Frodopir: Simple, scalable, single-server private information retrieval. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*, pages 365–383, 2023.

[48] Steve E. Deering. Host extensions for IP multicasting. RFC 1112, August 1989. URL: `https://www.rfc-editor.org/info/rfc1112`.

[49] Claudia Diaz, Harry Halpin, and Aggelos Kiayias. The Nym Network. *Self-published online*, 2021. Accessed September 2023. URL: `https://nymtech.net/nym-whitepaper.pdf`.

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

[51] European Commission DG Competition. Antitrust: Commission opens investigations into Apple's App Store rules, 2020. Accessed August 2023. URL: https://ec.europa.eu/commission/presscorner/detail/en/ip_20_1073.

[52] Freedom of the Press Foundation. Securedrop share and accept documents securely, 2023. Accessed September 2023. URL: https://securedrop.org/.

[53] Google Inc. Android O prevents access to /proc/stat , 2017. Accessed September 2023. URL: https://issuetracker.google.com/issues/37140047#comment2.

[54] Google Inc. Optimize for Doze and App Standby, 2021. Accessed September 2023. URL: https://developer.android.com/training/monitoring-device-state/doze-standby.

[55] Google Inc. Power management restrictions, 2021. Accessed September 2023. URL: https://developer.android.com/topic/performance/power/power-details.

[56] Google Inc. Hardware security best practices, 2022. Accessed September 2023. URL: https://source.android.com/docs/security/best-practices/hardware.

[57] Google Inc. Android and Google devices security reward program rules, 2022. Accessed September 2023. URL: https://bughunters.google.com/about/rules/6171833274204160/android-and-google-devices-security-reward-program-rules.

[58] Google Inc. Android Keystore system - hardware security module, 2022. Accessed September 2023. URL: https://developer.android.com/training/articles/keystore#HardwareSecurityModule.

[59] Google Inc. Android (Go edition) leveling up entry-level devices, 2023. Accessed August 2023. URL: https://www.android.com/versions/go-edition/.

[60] Matthew D Green and Ian Miers. Forward secure asynchronous messaging from puncturable encryption. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy (S&P '15)*, pages 305–320. IEEE, 2015.

[61] Alexandra Henzinger, Matthew M Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In *Proceedings of the 32nd USENIX Security Symposium (USENIX Security '23)*, volume 23, pages 3889–3905, 2023.

[62] Alex Hern. Instagram led users to Covid misinformation amid pandemic – report. *The Guardian*, 2021. Accessed January 2023. URL: https://www.theguardian.com/technology/2021/mar/09/instagram-led-users-to-covid-misinformation-amid-pandemic-report.

[63] Abram Hindle, Alex Wilson, Kent Rasmussen, E Jed Barlow, Joshua Charles Campbell, and Stephen Romansky. Greenminer: A hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 12–21, 2014.

[64] Mojtaba Hosseini, Dewan Tanvir Ahmed, Shervin Shirmohammadi, and Nicolas D Georganas. A survey of application-layer multicast protocols. *IEEE Communications Surveys & Tutorials*, 9(3):58–74, 2007.

[65] Junxian Huang, Feng Qian, Alexandre Gerber, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, pages 225–238, 2012.

[66] Daniel Hugenroth. Android Support for Elliptic Curves (EC) in KeyPairGenerator. Accessed September 2023. URL: `https://www.danielhugenroth.com/posts/2021_07_ec_curves_on_android`.

[67] Daniel Hugenroth and Alastair R Beresford. Powering privacy: On the energy demand and feasibility of anonymity networks on smartphones. In *Proceedings of the 32nd USENIX Security Symposium (USENIX Security '23)*, pages 5431–5448, 2023.

[68] Daniel Hugenroth, Martin Kleppmann, and Alastair R Beresford. Rollercoaster: An efficient group-multicast scheme for mix networks. In *Proceedings of the 30th USENIX Security Symposium (USENIX Security '21)*, pages 3433–3450, 2021.

[69] Daniel Hugenroth, Ceren Kocaoğullar, and Alastair R. Beresford. Choosing your friends: Shaping ethical use of anonymity networks. In *Proceedings of the Security Protocols XXVIII: 28th International Workshop*. Springer, 2023. Accepted and to be published.

[70] Daniel Hugenroth, Alberto Sonnino, Sam Cutler, and Alastair R. Beresford. Sloth: Key stretching and deniable encryption using secure elements on smartphones. 2023. Under Review.

[71] Claudia-Lavinia Ignat, Gérald Oster, Olivia Fox, Valerie L Shalin, and François Charoy. How do user groups cope with delay in real-time collaborative note taking. In *Proceedings of the 14th European Conference on Computer Supported Cooperative Work*, pages 223–242. Springer, September 2015.

[72] Internet Assigned Numbers Authority (IANN). Key Signing Ceremonies, 2023. Accessed September 2023. URL: `https://www.iana.org/dnssec/ceremonies`.

[73] Eric Jardine, Andrew M Lindner, and Gareth Owenson. The potential harms of the Tor anonymity network cluster disproportionately in free countries. *Proceedings of the National Academy of Sciences*, 117(50):31716–31721, 2020.

[74] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In *Proceedings of the Advances in Cryptology (EUROCRYPT)*, pages 456–486. Springer, 2018.

[75] Poul-Henning Kamp. GBDE—GEOM based disk encryption. In *BSDCon 2003 (BSDCon 2003)*, 2003.

[76] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.

[77] John Kelsey, Bruce Schneier, Chris Hall, and David Wagner. Secure applications of low-entropy keys. In *Proceedings of the Information Security: First International Workshop, (ISW '97)*, pages 121–134. Springer, 1997.

[78] Martin Kleppmann and Heidi Howard. Byzantine eventual consistency and the fundamental limits of peer-to-peer databases. *arXiv preprint arXiv:2012.00472*, 2020.

[79] Martin Kleppmann, Stephan A Kollmann, Diana A Vasile, and Alastair R Beresford. From secure messaging to secure collaboration. In *Proceedings of the Security Protocols XXVI: 26th International Workshop, Cambridge, UK*, pages 179–185. Springer, 2018.

[80] Martin Kleppmann, Adam Wiggins, Peter Van Hardenberg, and Mark McGranaghan. Local-first software: you own your data, in spite of the cloud. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pages 154–178, 2019.

[81] Stephan A Kollmann and Alastair R Beresford. The cost of push notifications for smartphones using Tor hidden services. In *Proceedings of the 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW '17)*, pages 76–85, 2017.

[82] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *CRYPTO*, volume 6223, pages 631–648. Springer, 2010.

[83] Christiane Kuhn, Martin Beck, and Thorsten Strufe. Breaking and (partially) fixing provably secure onion routing. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy (S&P '20)*, pages 168–185, 2020.

[84] Jacky Wei En Kung. Porting a mix network client to mobile, 2023. Part II Project.

[85] Ben Laurie. Certificate transparency. *Communications of the ACM*, 57(10):40–46, 2014.

[86] David Lazar and Nickolai Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, pages 571–586, 2016.

[87] Jinghui Liao, Bo Chen, and Weisong Shi. TrustZone enhanced plausibly deniable encryption system for mobile devices. In *Proceedings of the 2021 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 441–447. IEEE, 2021.

[88] Ewen MacAskill and Gabriel Dance. NSA files decoded: what the revelations mean for you. *The Guardian*, 2013. Accessed September 2023. URL: `https://www.theguardian.com/world/interactive/2013/nov/01/snowden-nsa-files-surveillance-revelations-decoded`.

[89] Andrew D McDonald and Markus G Kuhn. StegFS: A steganographic file system for Linux. In *Proceedings of the Information Hiding: Third International Workshop (IH '99)*, pages 463–477. Springer, 2000.

[90] Andrea McIntosh, Safwat Hassan, and Abram Hindle. What can Android mobile app developers do about the energy consumption of machine learning? *Empirical Software Engineering*, 24(2):562–601, 2019.

[91] Samir Jordan Menon and David J Wu. Spiral: Fast, high-rate single-server PIR via FHE composition. In *Proceedings of the 2022 IEEE Symposium on Security and Privacy (S&P '23)*, pages 930–947, 2022.

[92] José A Montenegro, Mónica Pinto, and Lidia Fuentes. What do software developers need to know to build secure energy-efficient Android applications? *IEEE Access*, 6:1428–1450, 2017.

[93] Kathleen Moriarty, Burt Kaliski, and Andreas Rusch. PKCS #5: Password-based cryptography specification version 2.1. RFC 8018, January 2017. URL: `https://www.rfc-editor.org/info/rfc8018`.

[94] Max Mössinger, Benedikt Petschkuhn, Johannes Bauer, Ralf C Staudemeyer, Marcin Wójcik, and Henrich C Pöhls. Towards quantifying the cost of a secure IoT: Overhead and energy consumption of ECC signatures on an ARM-based device. In *Proceedings of the 17th IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM '16)*, pages 1–6, 2016.

[95] Steven J Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (S&P '05)*, pages 183–195, 2005.

[96] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster protocol—version 2. *IETF Internet Draft*, 2003. Accessed September 2023. URL: `https://datatracker.ietf.org/doc/draft-sassaman-mixmaster/`.

[97] Petr Nalevka and Jiří Richter. Don't kill my app!, 2022. Accessed September 2023. URL: `https://dontkillmyapp.com/`.

[98] Jakob Nielsen. The 90-9-1 rule for participation inequality in social media and online communities, 2006. Accessed September 2023. URL: `https://www.nngroup.com/articles/participation-inequality/`.

[99] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. A survey of published attacks on Intel SGX. *arXiv preprint arXiv:2006.13598*, 2020.

[100] NYM Technologies SA. Nym Mainnet Explorer, 2023. Accessed June 2023. URL: `https://explorer.nymtech.net/`.

[101] NymTech. A Sphinx packet implementation in Rust, 2021. Accessed September 2023. URL: `https://github.com/nymtech/sphinx`.

[102] Open Worldwide Application Security Project (OWASP). Password storage cheat sheet, 2021. Accessed September 2023. URL: `https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html`.

[103] Abhinav Pathak, Y Charlie Hu, and Ming Zhang. Where is the energy spent inside my app? Fine grained energy accounting on smartphones with Eprof. In *Proceedings of the 7th ACM European Conference on Computer Systems*, pages 29–42, 2012.

[104] Kari Paul. We risk another crisis: TikTok in danger of being major vector of election misinformation. *The Guardian*, 2022. Accessed January 2023. URL: `https://www.theguardian.com/technology/2022/oct/24/tiktok-election-misinformation-voting-politics`.

[105] Colin Percival and Simon Josefsson. The scrypt password-based key derivation function. RFC 7914, August 2016. URL: `https://www.rfc-editor.org/info/rfc7914`.

[106] Trevor Perrin and Moxie Marlinspike. The Double Ratchet algorithm. 2016. Accessed September 2023. URL: `https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf`.

[107] Mike Perry and George Kadianakis. Tor padding specification, September 2021. Accessed September 2023. URL: `https://github.com/torproject/torspec/blob/main/padding-spec.txt`.

[108] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. *Self-published online*, 2010. Accessed September 2023. URL: `http://www.maroki.de/pub/dphistory/2010_Anon_Terminology_v0.34.pdf`.

[109] Sandro Pinto and Nuno Santos. Demystifying Arm TrustZone: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 51(6):1–36, 2019.

[110] Ania M Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The Loopix anonymity system. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security '17)*, pages 1199–1216, 2017.

[111] Adrian Popescu, Doru Constantinescu, David Erman, and Dragos Ilie. A survey of reliable multicast communication. In *Proceedings of the IEEE Conference on Next Generation Internet Networks*, NGI, pages 111–118, 2007.

[112] Nachiketh R Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K Jha. A study of the energy consumption characteristics of cryptographic algorithms and security protocols. *IEEE Transactions on Mobile Computing*, 5(2):128–143, 2005.

[113] VeraCrypt project. VeraCrypt - free open source disk encryption with strong security for the paranoid, 2023. `https://www.veracrypt.fr/en/Home.html`.

[114] Andrew Rice and Simon Hay. Measuring mobile phone energy consumption for 802.11 wireless networking. *Pervasive and Mobile Computing*, 6(6):593–606, 2010.

[115] Helena Rifà-Pous and Jordi Herrera-Joancomart. Computational and energy costs of cryptographic algorithms on handheld devices. *Future Internet*, 3(1):31–48, 2011.

[116] Keegan Ryan. Hardware-backed heist: Extracting ECDSA keys from Qualcomm's TrustZone. In *Proceedings of the 26th ACM Conference on Computer and Communications Security (ACM CCS '19)*, pages 181–194, 2019.

[117] Sajin Sasy and Ian Goldberg. SoK: Metadata-protecting communication systems. *Cryptology ePrint Archive*, 2023. Accessed September 2023. URL: `https://eprint.iacr.org/2023/313.pdf`.

[118] David Schatz, Michael Rossberg, and Guenter Schaefer. Hydra: Practical metadata security for contact discovery, messaging, and dialing. In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP '21)*, pages 191–203, 2021.

[119] Christoph Schnabl. Private group management for mix networks, 2023. Bachelor Thesis.

[120] Alon Shakevsky, Eyal Ronen, and Avishai Wool. Trust dies in darkness: Shedding light on Samsung's TrustZone Keymaster design. In *Proceedings of the 31st USENIX Security Symposium (USENIX Security '22)*, pages 251–268, 2022.

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

[122] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS '11)*, pages 386–400. Springer, 2011.

[123] Laurent Simon, Wenduan Xu, and Ross Anderson. Don't interrupt me while I type: Inferring text entered through gesture typing on Android keyboards. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2016.

[124] Adam Skillen and Mohammad Mannan. Mobiflage: Deniable storage encryption for mobile devices. *IEEE Transactions on Dependable and Secure Computing*, 11(3):224–237, 2013.

[125] statcounter GlobalStats. Mobile operating system market share, 2023. Accessed August 2023. URL: `https://gs.statcounter.com/os-market-share/mobile/`.

[126] Texas Instruments. *INA219 Zero-Drift, Bidirectional Current/Power Monitor with I2C Interface (SBOS448G)*, 12 2015. Rev. G. URL: `https://www.ti.com/lit/ds/symlink/ina219.pdf`.

[127] The Economist. How social-media platforms dispense justice. *The Economist*, 2018. Accessed January 2023. URL: `https://web.archive.org/web/20190516115755/https://www.economist.com/business/2018/09/06/how-social-media-platforms-dispense-justice`.

[128] The Guardian Project. Orbot: Proxy with Tor, 2022. Accessed September 2023. URL: `https://guardianproject.info/apps/org.torproject.android/`.

[129] The International Telecommunication Union. Internet surge slows, leaving 2.7 billion people offline in 2022, 2022. Accessed August 2023. URL: `https://www.itu.int/en/mediacentre/Pages/PR-2022-09-16-Internet-surge-slows.aspx`.

[130] The International Telecommunication Union. Regional and global key ICT indicator, 2023. Accessed August 2023. URL: https://www.itu.int/en/ITU-D/Statistics/Documents/facts/ITU_regional_global_Key_ICT_indicator_aggregates_Nov_2022_revised_15Feb2023.xlsx.

[131] The Tor Project. Tor user metrics, 2023. Accessed May 2023. URL: https://metrics.torproject.org/userstats-relay-country.html.

[132] Mehdi Tibouchi. Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings. In *Proceedings of the International Conference on Financial Cryptography and Data Security*, pages 139–156. Springer, 2014.

[133] Tor Project. Abuse FAQ - Doesn't Tor enable criminals to do bad things?, 2023. Accessed February 2023. URL: https://support.torproject.org/abuse/what-about-criminals/.

[134] Robert Triggs. Fact check: Is smartphone battery capacity growing or staying the same? , 2018. Accessed September 2023. URL: https://www.androidauthority.com/smartphone-battery-capacity-887305.

[135] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the Proceedings of the 26th Symposium on Operating Systems Principles*, pages 423–440, 2017.

[136] United Nations Committee for Development Policy. The Least Developed Countries category: countries snapshots, 2021. Accessed August 2023. URL: https://www.un.org/development/desa/dpad/wp-content/uploads/sites/45/Snapshots2021.pdf.

[137] United Nations Committee for Development Policy. List of Least Developed Countries, 2023. Accessed August 2023. URL: https://www.un.org/development/desa/dpad/wp-content/uploads/sites/45/publication/ldc_list.pdf.

[138] United Nations Population Division. Database on household size and composition, 2022. Accessed August 2023. URL: https://www.un.org/development/desa/pd/sites/www.un.org.development.desa.pd/files/undesa_pd_2022_hh-size-composition.xlsx.

[139] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152, 2015.

[140] Matteo Varvello, Kleomenis Katevas, Mihai Plesa, Hamed Haddadi, and Benjamin Livshits. BatteryLab: a distributed power monitoring platform for mobile devices. In *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets '19)*, 2019.

[141] Diana-Alexandra Vasile. *Securing encrypted communication.* PhD thesis, University of Cambridge, 2023.

[142] Ekhiotz Jon Vergara, Simon Andersson, and Simin Nadjm-Tehrani. When mice consume like elephants: Instant messaging applications. In *Proceedings of the 5th International Conference on Future Energy Systems*, pages 97–107, 2014.

[143] Ekhiotz Jon Vergara, Simin Nadjm-Tehrani, and Mihails Prihodko. EnergyBox: Disclosing the wireless transmission energy cost for mobile devices. *Sustainable Computing: Informatics and Systems*, 4(2):118–135, 2014.

[144] Daniel T Wagner, Andrew Rice, and Alastair R Beresford. Device analyzer: Understanding smartphone usage. In *Proceedings of the International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 195–208. Springer, 2013.

[145] Tao Wang and Ian Goldberg. Improved website fingerprinting on Tor. In *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society (WPES '13')*, pages 201–212, 2013.

[146] Matthew Weidner, Martin Kleppmann, Daniel Hugenroth, and Alastair R Beresford. Key agreement for decentralized secure group messaging with strong security guarantees. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (ACM CCS '21)*, pages 2024–2045, 2021.

[147] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*, pages 179–182, 2012.

[148] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*, pages 179–182, 2012.

[149] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Scalable anonymous group communication in the anytrust model. In *Proceedings of the European Workshop on System Security (EuroSec)*, volume 4, 2012.

[150] Chai Kiat Yeo, Bu-Sung Lee, and Meng H Er. A survey of application level multicast techniques. *Computer Communications*, 27(15):1547–1568, 2004.

[151] Bassam Zantout, Ramzi Haraty, et al. I2P data communication system. In *Proceedings of the 10th International Conference on Networks (ICN)*, pages 401–409. ICN, Springer Singapore, 2011.

[152] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 105–114, 2010.

Acronyms and abbreviations

ACK	Acknowledgement (message).
AE	Authenticated encryption.
AES	Advanced encryption standard.
AOSP	Android open-source project.
API	Application programming interface.
ASIC	Application-specific integrated circuit.
CCA	Chosen-ciphertext attack.
CDN	Content delivery network.
CPA	Chosen-plaintext attack.
CRDT	Conflict-free replicated data type.
CSV	Comma-separated values.
DCGKA	Decentralized continuous group key agreement.
DDH	Decisional Diffie–Hellman.
DEMS	Deniable encryption multi-snapshot.
DESS	Deniable encryption single-snapshot.
DH	Diffie–Hellman.
DNSSEC	Domain name system security extensions.
DoS	Denial of service.
E^2	Elligator Squared.
E2EE	End-to-end encrypted (or encryption).
EC	Elliptic curve.
ECC	Elliptic curve cryptography.
ECDH	Elliptic curve Diffie-Hellman.
EXP	Exponential (distribution).
FIFO	First-in first-out.
FT	Fault-tolerance.

GiB	Gibibyte (2^{30} Bytes).
GNU	GNU's not Unix.
GPA	Global passive adversary.
GPS	Global positioning system.
HAL	Hardware abstraction layer.
HIBE	Hierarchical identity-based encryption.
HKDF	HMAC-based key derivation function.
HMAC	Hash-based message authentication code.
HTTPS	Hypertext transfer protocol secure.
I2P	Invisible Internet Project.
IDE	Integrated development environment.
IND	Indistinguishability.
IP	Internet protocol.
ISP	Internet service provider.
ITU	International Telecommunication Union.
JNI	Java native interface.
KDF	Key derivation function.
KiB	Kibibyte (2^{10} Bytes).
LDCs	Least developed countries.
MAC	Message authentication code.
MiB	Mebibyte (2^{20} Bytes).
OS	Operating system.
OWASP	Open worldwide application security project.
P2P	Peer-to-peer.
PAKE	Password-authenticated key exchange.
PDA	Personal digital assistant.
PE	Puncturable encryption.
PET	Privacy enhancing technology.
PGM	Private group management.
PIR	Private information retrieval.
POIS	Poisson (distribution).
PP	Percentage points.

PPT	Probabilistic polynomial-time.
PRF	Pseudo-random function.
PRNG	Pseudo-random number generator.
PSQ	PrivateSendingQueue.
RC	Rollercoaster.
SE	Secure element.
SGX	Intel Software Guard Extensions.
SIM	Subscriber identity module.
SSL	Secure sockets layer.
TCP	Transmission control protocol.
TEE	Trusted execution environment.
TLS	Transport layer security.
UDP	User datagram protocol.
UN	United Nations.
VPN	Virtual private network.
WT	Wall-time.
WTA	Wall-time adversary (or algorithm).

Typography note: We typeset the introduced protocols and operations using SmallCaps in this dissertation. We use *italics* for emphasising terms and expressions.

Appendix A

Rollercoaster

A.1 Algorithms

Algorithm 8 Methods explaining how the timeout information is stored and updated.

```
1:  procedure OnInit
2:      self.sessions = [·]                          ▷ missing keys default to {}
3:
4:  procedure AddTimeout(msg, role, recipient, timeout)
5:      CancelTimeout(msg, role, recipient)
6:      id ← (msg.groupid, msg.nonce)
7:      entry ← (role, recipient, timeout)
8:      self.sessions[id] ← self.sessions[id] ∪ {entry}
9:
10: procedure CancelTimeout(msg, role, recipient)
11:     id ← (msg.groupid, msg.nonce)
12:     session = self.sessions[id]
13:     self.sessions[id] ← {x ∈ self.sessions[id] | x.role ≠ role ∧ x.recipient ≠ recipient}
```

Algorithm 9 Determines whether node *node* is a forwarding node with regards to schedule S.

```
1:  procedure IsForwardingNode(S, node)
2:      source ← S[0][0][0]
3:      if node = source then
4:          return false
5:      for t = 1 until |S| do
6:          R ← S[t]
7:          for (sender, _) in R do
8:              if node ≠ source and node = sender then
9:                  return true
10:     return false
```

Algorithm 10 The fault-tolerant ROLLERCOASTER callback handler and send methods (signatures are checked implicitly).

```
 1: procedure SendToGroup(groupid, payload)
 2:     S ← GenSchedule(msg.source, msg.groupid)
 3:     for recipient ∈ {direct children of self in S} do
 4:         msg ← NewMessage()
 5:         msg.groupid ← groupid
 6:         msg.nonce ← FreshNonce()
 7:         msg.{source, sender, role} ← self
 8:         msg.payload ← payload
 9:         ScheduleForSend(recipient, msg)
10:
11: procedure OnPayload(msg)
12:     ApplicationHandle(msg.payload)
13:     if msg was received while offline then return
14:     if msg was not seen before then
15:         S ← GenSchedule(msg.source, msg.groupid)
16:         for x ∈ {direct children of msg.role in S} do
17:             msg′ ← CopyMessage(msg)
18:             msg′.sender ← self
19:             msg′.role ← x
20:             ScheduleForSend(x, msg′)
21:     ScheduleForSend(msg.source, GenAck(msg))
22:
23: procedure OnAck(msg)
24:     assert (msg.source = self)
25:     CancelTimeout(msg, msg.role, msg.sender)
26:
27: procedure OnMessageIsSent(msg)        ▷ Called when a message leaves the payload queue
28:     S ← GenSchedule(msg.source, msg.groupid)
29:     for x ∈ {recursive children of msg.role in S} do
30:         timeout ← EstimateTimeout(S, x)
31:         AddTimeout(msg, x, timeout)
32:
33: procedure OnTimeout(msg, recipient_failed)
34:     S ← GenSchedule(msg.source, msg.groupid)
35:     if not IsForwardingNode(S, msg.role) then
36:         return
37:     for x ∈ {recursive children of msg.role in S} do
38:         CancelTimeout(msg, msg.role, msg.sender)
39:                                          ▷ timeout will be recreated when re-try is sent
40:     recipient′ ← NextRecipient(S, recipient_failed)
41:     ScheduleForSend(recipient′, msg)
42:     msg.role ← ∅                                  ▷ Re-try to failed node w/o role
43:     ScheduleWithExpBackoff(recipient_failed, msg)
```

Algorithm 11 Determines the next recipient after discovering a faulty node f given schedule S

```
1: procedure NextRecipient(S, f)
2:     order ← [S[0][0][0]]                          ▷ start with the source
3:     for t = 0 until |S| do
4:         R ← S[t]
5:         for (_, recipient) in R do
6:             if recipient ∉ order then
7:                 order.append(recipient)
8:     pos ← order.findIndex(f)
9:     return order[(pos + 1) mod |order|]
```

Intentionally left empty.

A.2 Heatmaps

In Figure A.1 we present additional online and offline scenarios for different sending rates and branching factors without (left) and with (right) p-restricted multicast. See Section 4.4.5.

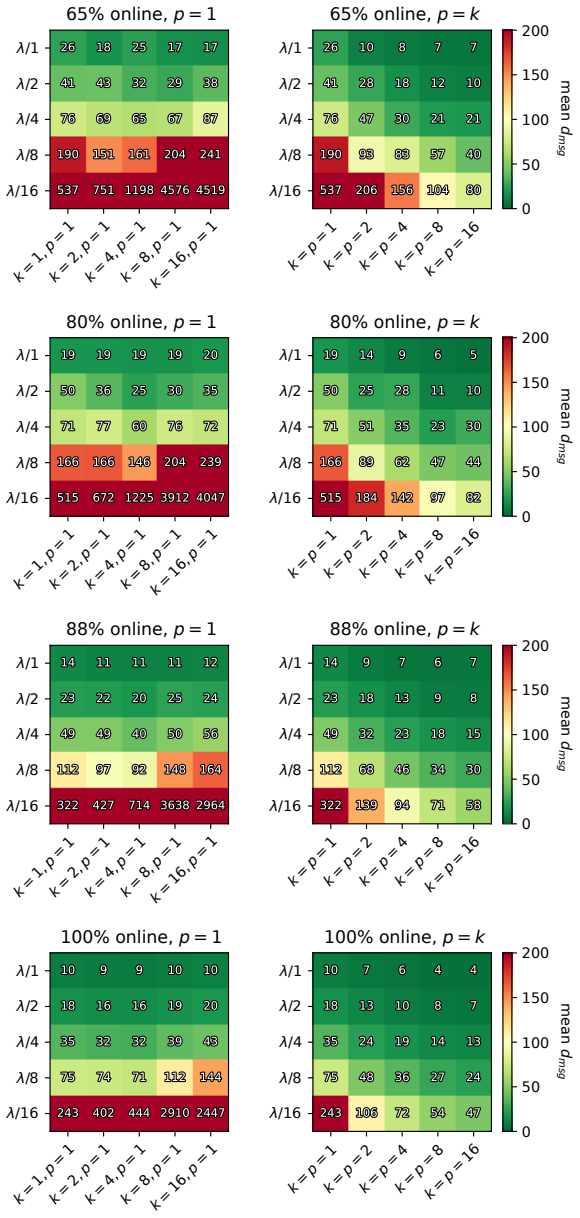

Figure A.1: Heatmaps showing the mean message latency for reduced sending rates (y-axis) and different Rollercoaster parameters (x-axis). In the left graph only the logical multiplier factor k is increased. In the right graph the multicast factor p is increased at the same time. Group size 128.

A.3 Histograms

Figures A.2 and A.3 (next page) provide additional illustrations and express the underlying behaviour of the different strategies. They serve as additional evidence that our simulator functions as described. The histograms also show the results that are provided in the box charts in Figures 4.5 and 4.6.

The distribution of the online naïve sequential unicast strategy in Figure A.2 has a very wide body that is an effect of the queuing time in the payload buffer of the sender. Its height depends on the payload rate λ_p. On the other hand, the ROLLERCOASTER graphs show a much tighter distribution with a much higher peak (124k compared to 11k for unicast).

For the offline scenarios in Figure A.3 the distribution for the naïve sequential unicast strategy is almost the same as in Figure A.2. The non-fault tolerant ROLLERCOASTER strategy fails dramatically and is dominated by extreme outliers – with p_{99} growing to multiple hours! Adding the basic fault tolerance improves the p_{90} percentile and the mean values approach a reasonable order of magnitude. However, nodes coming back online continue to be overwhelmed since they first work through old messages in their mailbox. The fix, as described in the main text, is to drop any forwarding messages received while the node was offline. Doing so means that the fault-tolerant ROLLERCOASTER strategy provides excellent results with mean, p_{90}, and p_{99} being less than a minute and better than unicast can provide. The individual peaks visible are the result of timeouts and subsequent re-transmissions.

Intentionally left empty.

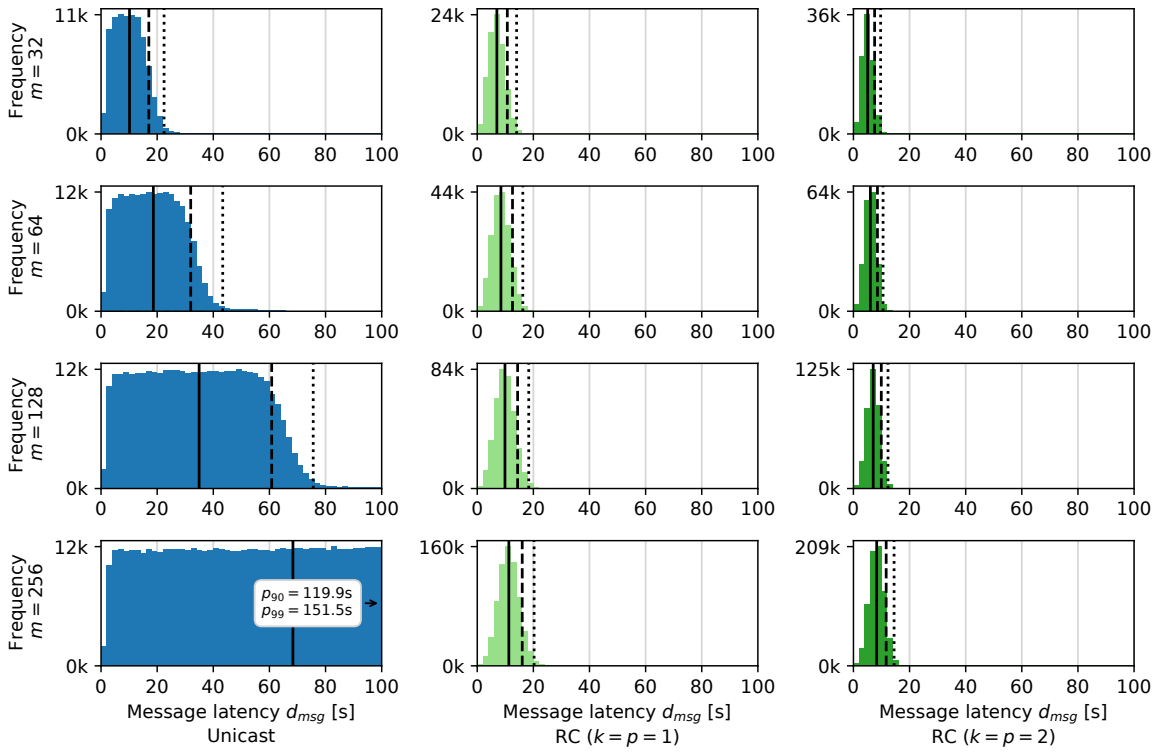

Figure A.2: The distribution of message latency of naïve sequential unicast and Rollercoaster (RC) as perceived by the participating group members in a simulation where all users are online all the time. The solid line marks the mean latency whereas the dashed (and dotted) lines mark the p_{90} (and p_{99}) latency. In this figure the y-axes are not linked in order to provide higher fidelity.

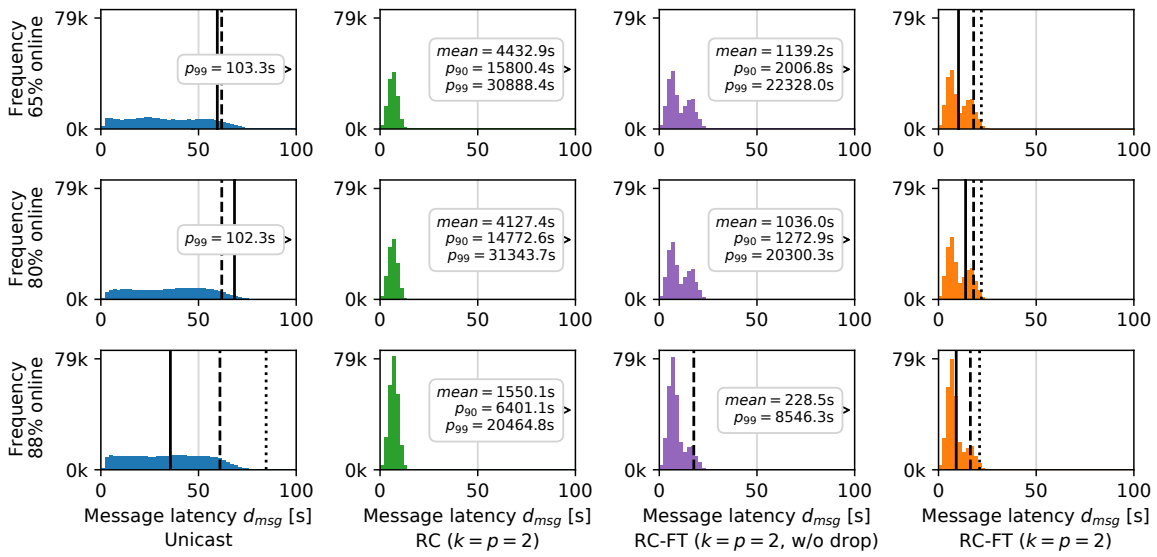

Figure A.3: The distribution of message latency d_{msg} for sequential unicast and different RollerCoaster configurations. The group size is 128 and the rows show different offline scenarios. The left graphs show (left-to-right): (i) Naïve unicast, (ii) RollerCoaster without fault-tolerance for $k = p = 2$, (iii) RollerCoaster with fault-tolerance but not ignoring messages that arrived while offline, (iv) our final RollerCoaster algorithm with fault-tolerance and all optimisations. The solid line marks the mean latency whereas the dashed (and dotted) lines mark the p_{90} (and p_{99}) latency. The mean can be larger than the p_{90} if there are few but large outliers.

A.4 Eventual delivery proof

We make the following assumptions for the remainder of Appendix A.4: All network links are fair-loss and deliver messages with probability > 0. Every layer of the mix network has at least one mix node that correctly forwards messages. Any node can go offline at any time and all nodes except *source* can be Byzantine-faulty. We use the following definitions for the remainder of this section: Let *source* be a node, which does not go offline permanently. Let S be a schedule that includes *source* as its root and all group members as its internal and leaf nodes.

Lemma 23. *Every message which is sent along a randomly chosen path of mix nodes has probability* > 0 *to be delivered.*

Proof. Since there is a mix node that correctly forwards messages in every layer, there is a non-zero probability of choosing mix nodes in all three layers which all correctly forward messages. Since every network link has a non-zero probability of delivering a message, the entire route consisting of multiple network links has a non-zero probability of delivering the message. Therefore, sending a message along a randomly chosen path has probability > 0 of being delivered. □

Lemma 24. *Every direct payload sent by a node* sender*, which does not go offline permanently, to a node* recipient *is eventually delivered.*

Proof. We first consider the case (a) that the delivery of the payload to *recipient* is successful and the delivery of the ACK to *sender* is successful. In this case *sender* can be certain that the payload was delivered to *recipient*, because only *recipient* can compute the correct signature in the ACK. Therefore, the cancellation of the timeout by *sender* is safe.

We now consider the case (b) that the delivery of the payload to *recipient* failed or the delivery of the ACK to *sender* failed. Any of the two failures causes the timeout at the *sender* to expire. As a result *sender* sends the same payload again using a new random path and setting a new timeout. Since sending the payload to *recipient* and sending the ACK to *sender* have non-zero probability of success (Lemma 23), there will eventually be an execution where both succeed.

We now consider the case (c) that *recipient* is offline. For *sender* this situation is indistinguishable from case (b). Therefore, *sender* will retry until *recipient* returns online and sends an ACK.

We now consider the case (d) that *sender* is offline. If the payload has not been sent yet, it will be sent when *sender* returns online. In case (a) *sender* will observe the ACK message when it returns online. In cases (b) and (c) the sender will re-try once it returns online and the timeout expired. Since *sender* will not go offline permanently, transition to cases (a)-(c) will happen eventually. □

Lemma 25. *Any payload sent by* source *is eventually delivered to all direct children of* source *in* S*.*

Proof. For direct children of *source* in S, all payloads are sent as direct messages. From Lemma 24 it follows that these are delivered eventually. □

Lemma 26. *Any payload sent by* source *is eventually delivered to all indirect children of* source *in* S*.*

Proof. Let x be an arbitrary indirect child of *source* in S.

We first consider the case (a) that the payload was delivered through the forwarding node(s) to x and the ACK message was delivered to *source*. It is safe for *source* to cancel the respective timeout, as the payload was delivered to x.

We now consider the case (b) that x is a direct child of p which is a direct child of *source*. If *source* receives p's ack, but not x's ack, then x becomes a direct child of *source*. This case is considered in Lemma 25. Otherwise, p timed-out before x, because p's timeout is strictly smaller than the one of x. This means that *source* assigns the role of p to another node p' and sets new timeouts for p' and x. If p' fails, this is repeated for multiple rounds. The list of replacement nodes shrinks by ≥ 1 every round as previously failed nodes will not be considered again. Therefore, eventually forwarding succeeds or *source* (which is part of the replacement list) becomes p' making x a direct recipient. This case is considered in Lemma 25.

We now consider the case (c) that x is a child in a tree path $[source, p_1, p_2, \ldots, x]$. Let p_j with $j \geq 1$ be the parent closest to x whose timeout expires. This implies that all p_i with $i < j$ successfully acknowledged to *source* since their timeouts are strictly smaller. The role of p_j will be assigned to another node p'_j and new timeouts are set. If p'_j fails, this is repeated for multiple rounds. Therefore, eventually forwarding succeeds or *source* becomes p'_j which reduces the length of the path from *source* to x by at least one. Therefore, eventually forwarding succeeds or the path length is reduced so far that case (b) applies. □

Theorem 27. *Let* source *be a node that does not go offline permanently, and that is not Byzantine-faulty. All payloads sent by* source *are eventually delivered to all group members.*

Proof. For any schedule, every group member is either a direct child or an indirect child of *source*. Therefore, the result follows directly from Lemma 25 and Lemma 26. □

Theorem 28. *Let* source′ *be a node that may be Byzantine-faulty. Let X be any subset of group members that are not Byzantine-faulty and that do not go offline permanently. If one of the nodes in X receives a payload from* source′*, all other nodes in X will eventually receive the payload.*

Proof. Let $x \in X$ be the node that received a payload message m from *source*′, and let x' be any other member of X. We then show that x' eventually receives m.

As described in Section 4.3.2, every node periodically computes a hash of the payloads it received and sends it to a randomly selected group member; thus, x eventually computes a hash over a message history including m and sends it to x', and by Lemma 23 this message is eventually received by x' (possibly after several attempts). If x' has already received m, we are done. If x' has not yet received m, and assuming the hash function is collision-resistant, then there is no message history known to x' that results in the same hash value, and therefore the hash sent by x is unknown to x'.

x' responds to the unknown hash by sending a request to x, asking it to send any messages that x' is missing. A simple but inefficient algorithm would be for x to resend all payload messages it has ever received to x'. A more efficient approach uses a reconciliation protocol to determine which messages are known to x but unknown to x', and to resend only those messages. Several such reconciliation protocols are known [78]. Whatever protocol is used, it will eventually complete (by Lemma 23), and therefore x' will eventually receive m from x, as required. □

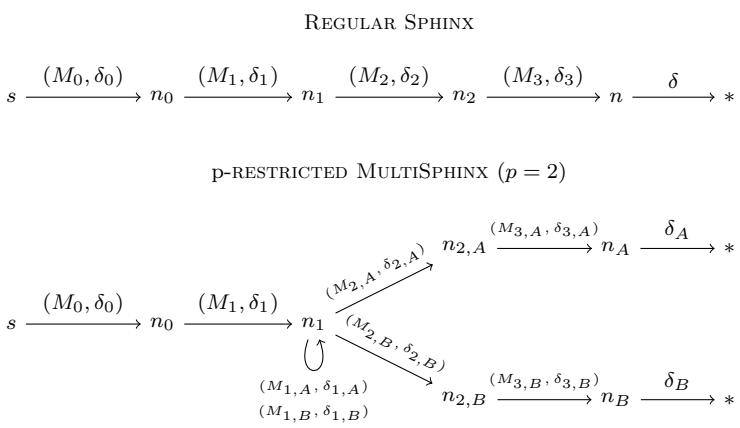

Figure A.4: Schematic of messages (header, payload) for Sphinx and MultiSphinx.

A.5 MultiSphinx construction

In this Appendix we provide detailed algorithms for constructing and processing both the regular Sphinx messages (A.5.1) and our MultiSphinx messages (A.5.2). The regular construction is based on the original Sphinx paper [44] and the proposed improvement using authenticated encryption [20]. For both schemes we will use three hops n_0, n_1, n_2 for the mix nodes and a final hop n for the recipient[1] that extracts the payload from the inner-most encryption (Figure A.4).

A Sphinx header M consists of a group element α for deriving shared secrets, authenticated data β, and an authentication tag γ. In the original Sphinx paper β is used to store the address of the next hop. For the final hop the distinguished element $*$ is used to signal that the payload reached its intended destination. Loopix adds per-hop delays to this routing information.

We assume that all nodes n_i have access to the public keys of all other nodes without us passing these explicitly. We assume the existence of a method ProcessHeader that takes a header of a Sphinx packet and returns all metadata contained in β (next hop identifier, delay) and the header for the next hop. We assume the existence of a method ComputeSecrets that takes a list of hops $n_0, n_1, \ldots$ and outputs a list of shared secrets $s_0, s_1, \ldots$. We assume the existence of a method CreateHeader that takes a shared secret s_i, the next hop identifier n_{i+1}, and (optionally) a header M_{i+1} to wrap. The details of these operations can be found in the Sphinx paper [44, §3.2 and §3.6]. In line with Loopix the sender chooses a random per-hop delay for each hop and includes it in the authenticated metadata in the header. This happens transparently in the CreateHeader method.

We assume the existence of an authenticated encryption (AE) scheme as required by the improved Sphinx format [20]. An AE scheme provides an encryption function AE_{enc} that takes a secret key s, a message msg, and optional metadata $meta$ and outputs a ciphertext $ctext$ and an authentication tag $auth$. It also provides a decryption function AE_{dec} that takes a secret key s, a ciphertext $ctext$, an authentication tag $auth$, and metadata $meta$. It returns the decrypted message if the authentication tag verifies the integrity of ciphertext and metadata or $\perp$ otherwise.

We assume that the AE scheme is based on an encrypt-then-mac regime using a stream cipher C (e.g. AES-CTR), a message authentication code MAC (e.g. HMAC), and a keyed key derivation

[1] We omit the provider nodes here to improve readability.

function KDF (e.g. HKDF). Stream ciphers have the property that changing a given bit of the ciphertext/plaintext only changes the bit at the same position in the plaintext/ciphertext after decryption/encryption. Arbitrary changes will lead to an invalid *auth* tag – but we might intentionally ignore this during our constructions and recalculate the *auth* tags later. Since Sphinx uses fresh secret keys for every message and hop, we can leave the nonce for the stream cipher constant. We show our construction of AE_{enc} and AE_{dec} in Algorithm 12.

Algorithm 12 The authenticated encryption scheme AE based on stream cipher C, a MAC, and a keyed KDF.

procedure $AE_{enc}(s, msg, meta)$
  $s_{cipher}, s_{mac} \leftarrow \text{KDF}(s, \texttt{cipher}), \text{KDF}(s, \texttt{mac})$
  $ctext \leftarrow \text{C}(s_{cipher}) \oplus msg$
  $auth \leftarrow \text{MAC}(s_{mac}, ctext \parallel meta)$
  return $(ctext, auth)$

procedure $AE_{dec}(s, ctext, auth, meta)$
  $s_{cipher}, s_{mac} \leftarrow \text{KDF}(s, \texttt{cipher}), \text{KDF}(s, \texttt{mac})$
  if $\text{MAC}(s_{mac}, ctext \parallel meta) \neq auth$ **then**
    return $\perp$
  $msg \leftarrow \text{C}(s_{cipher}) \oplus ctext$
  return msg

A.5.1 Normal Sphinx (existing solution)

The algorithms in this section summarise the existing literature [20, 44], but we adapted the notation to be more concise. Algorithm 13 shows the creation of the a regular Sphinx message by the sender. While the original Sphinx papers can create all headers before encrypting the payload, the improved variant with AE requires us to do these operations simultaneously as the encryption affects the authentication tag γ of this and the following message headers.

Algorithm 13 Creating a packet to be routed through hops n_0, n_1, n_2 to node n.

procedure CREATE$(\delta, n_0, n_1, n_2, n)$
  assert$|\delta| =$ MAXMSGLEN
  $s_0, s_1, s_2, s_3 \leftarrow$ COMPUTESECRETS(n_0, n_1, n_2, n)
  $M_3 \leftarrow$ CREATEHEADER$(s_3, *)$
  $\delta_3,\ M_3.\gamma \leftarrow AE_{enc}(s_3, \delta, M_3.\beta)$
  $M_2 \leftarrow$ CREATEHEADER(s_2, n, M_3)
  $\delta_2,\ M_2.\gamma \leftarrow AE_{enc}(s_2, \delta_3, M_2.\beta)$
  $M_1 \leftarrow$ CREATEHEADER(s_1, n_2, M_2)
  $\delta_1,\ M_1.\gamma \leftarrow AE_{enc}(s_1, \delta_2, M_1.\beta)$
  $M_0 \leftarrow$ CREATEHEADER(s_0, n_1, M_1)
  $\delta_0,\ M_0.\gamma \leftarrow AE_{enc}(s_0, \delta_1, M_0.\beta)$
  return (M_0, δ_0)

Algorithm 14 shows how a mix node processes a message it received. First the message is unpacked into the header and the payload. Then the tag is derived and compared against previously seen *tags* to protect against replay attacks. Afterwards, the decryption verifies that the authentication tag

matches the message and header metadata. Finally the header is unwrapped and a send operation is scheduled according to the next hop identifier and delay from the metadata.

Algorithm 14 Processing of an incoming packet at mix node n with secret key x_n.

1: **procedure** PROCESS($packet$)
2: $(M, \delta) \leftarrow packet$
3: $s \leftarrow (M.\alpha)^{x_n}$
4: **if** $h_\tau(s) \in tags$ **then abort**
5: $tags \leftarrow tags \cup \{h_\tau(s)\}$
6: $\delta' \leftarrow \mathrm{AE}_{dec}(s, \delta, M.\gamma, M.\beta)$
7: **if** $\delta' = \perp$ **then abort**
8: $(n', delay), M' = \text{PROCESSHEADER}(M)$
9: QUEUEFORSEND$(n', (M', \delta'), delay)$

A.5.2 MultiSphinx (our solution)

We now describe our MULTISPHINX construction and highlight the changes relative to the normal Sphinx construction in blue. To allow for a readable description we describe everything for $p = 2$ however the general case follows easily.

We use the pseudo-random function (PRF) ρ together with its key-generating function h_ρ from the original Sphinx paper to create a deterministic pseudo-random padding. Since we need two derive to independent keys from the same secret, we extend h_ρ with another parameter that can be an arbitrary string. This extension can be implemented using any suitable HKDF function.

Algorithm 15 explains the creation of MULTISPHINX messages by the sender. The part concerning the "two legs" of the message graph is only shown once for `A` to allow for a more readable presentation. Line 21 instructs which lines are meant to be repeated for the other $p - 1$ recipients. In line 4 the secret s_1 is computed which is required for the padding construction in line 11. Lines 6-9 encrypt the actual payload from the recipient n_A to the multiplication node $n_{1,A}$ (going backwards). The encrypted payloads $\delta_{3,A}, \delta_{2,A}, \delta_{1,A}$ are all smaller than the normal payload length of messages. This would allow an attacker to distinguish such messages from other Loopix messages (e.g. when the middle mix layer sends loop messages). Therefore, the ciphertext is padded in line 11 with our PRF ρ. To correctly compute the MACs and headers in lines 15-20, we first simulate (going forwards) how the payloads will be affected by the decryption (line 12f).

Algorithm 16 explains the processing step at a mix node. Regular mix nodes operate as before (line 10). However, at multiplication nodes incoming message payloads are split into p headers and p payloads (line 12). In lines 13-16 the pseudo-random paddings are added. This process is also visualised in Figure 4.4. The newly created packets are processed recursively and then scheduled for sending based on their individual delay (line 15f). This "self-delivery" corresponds to the loop edge of n_1 in Figure A.4. The extra hop allows for delaying both messages independently at the multiplication node (two headers allow for two delays). It also simplifies our correctness arguments.

Algorithm 15 Creating a MultiSphinx packet to be routed through hops $n_0, n_1, n_{2,A}, n_{2,B}$ to nodes n_A, n_B.

1: **procedure** CREATE($\delta_A, \delta_B, n_0, n_1, n_{2,A}, n_{2,B}, n_A, n_B$)
2: **assert**$|\delta_A| = |\delta_B| = (\text{MAXMSGLEN} - \text{HDRLEN})/2$
3: $s_0, s_1, \leftarrow$ COMPUTESECRETS(n_0, n_1) ▷ Secrets for hops from sender to multiplier node n_1
4:
5: ▷ Encrypt from recipient n_A to multiplier node n_1
6: $s_{1,A}, s_{2,A}, s_A \leftarrow$ COMPUTESECRETS$(n_{1,A}, n_{2,A}, n_A)$
7: $\delta_{3,A} \leftarrow \text{C}(\text{KDF}(s_A, \texttt{cipher})) \oplus \delta_A$
8: $\delta_{2,A} \leftarrow \text{C}(\text{KDF}(s_{2,A}, \texttt{cipher})) \oplus \delta_{3,A}$
9: $\delta_{1,A} \leftarrow \text{C}(\text{KDF}(s_{1,A}, \texttt{cipher})) \oplus \delta_{2,A}$
10:
11: ▷ Add pseudo-random padding and compute padded payloads $\delta'_{\dots}$ along decryption path
12: $\delta'_{1,A} \leftarrow \delta_{1,A} \parallel \rho(h_\rho(\texttt{A}, s_1))$
13: $\delta'_{2,A} \leftarrow \text{C}_{dec}(\text{KDF}(s_{1,A}, \texttt{cipher})) \oplus \delta'_{1,A}$
14: $\delta'_{3,A} \leftarrow \text{C}_{dec}(\text{KDF}(s_{2,A}, \texttt{cipher})) \oplus \delta'_{2,A}$
15:
16: $M_{3,A} \leftarrow$ CREATEHEADER$(s_A, *)$ ▷ Compute headers and full MACs
17: $M_{3,A}.\gamma \leftarrow \text{MAC}(\text{KDF}(s_A, \texttt{mac}), \delta'_{3,A} \parallel M_{3.A}).\beta)$
18: $M_{2,A} \leftarrow$ CREATEHEADER$(s_A, n_{3,A}, M_{3,A})$
19: $M_{2,A}.\gamma \leftarrow \text{MAC}(\text{KDF}(s_{2,A}, \texttt{mac}), \delta'_{2,A} \parallel M_{2.A}).\beta)$
20: $M_{1,A} \leftarrow$ CREATEHEADER$(s_A, n_{2,A}, M_{2,A})$
21: $M_{1,A}.\gamma \leftarrow \text{MAC}(\text{KDF}(s_{1,A}, \texttt{mac}), \delta'_{1,A} \parallel M_{1.A}).\beta)$
22: **Repeat lines** $6-20$ **for** B
23:
24: $\delta_{combined} = M_{1,A} \parallel \delta_{1,A} \parallel M_{1,B} \parallel \delta_{1,B}$ ▷ From sender to multiplication node
25: $M_1 \leftarrow$ CREATEHEADER(s_1)
26: $\delta_1,\ M_1.\gamma \leftarrow AE_{enc}(s_1, \delta_{combined}, M_1.\text{METADATA}))$
27: $M_0 \leftarrow$ CREATEHEADER(s_0, n_1, M_1)
28: $\delta_0,\ M_0.\gamma \leftarrow AE_{enc}(s_0, \delta, M_0.\text{METADATA}))$
29: **return** (M_0, δ_0)

Algorithm 16 Processing of an incoming packet at mix node n at mix layer l with secret key x^n.

1: **procedure** PROCESS($packet$)
2: $(M, \delta) \leftarrow packet$
3: $s \leftarrow (M.\alpha)^{x_n}$
4: **if** $h_\tau(s) \in tags$ **then abort**
5: $tags \leftarrow tags \cup \{h_\tau(s)\}$
6: $\delta' \leftarrow AE_{dec}(s, \delta, M.\gamma, M.\beta)$
7: **if** $\delta' = \perp$ **then abort**
8: $n', delay, M' =$ PROCESSHEADER(M)
9: **if** $l \neq 1$ **then**
10: QUEUEFORSEND$(n', (M', \delta'), delay)$
11: **else**
12: $M_{1,A}, \delta_{1,A}, M_{1,B}, \delta_{1,B} \leftarrow \delta'$ ▷ $\delta' = \delta_{combined}$
13: $\rho_A, \rho_B \leftarrow \rho(h_\rho(\texttt{A}, s)), \rho(h_\rho(\texttt{B}, s))$ ▷ $s = s_1$
14: ▷ Process separately to allow independent delays
15: PROCESS$(M_{1,A} \parallel \delta_{1,A} \parallel \rho_A)$
16: PROCESS$(M_{1,B} \parallel \delta_{1,B} \parallel \rho_B)$

A.6 MultiSphinx proofs

This appendix provides proofs for the MultiSphinx security and anonymity claims in Section 4.3.4. We discuss the resistance against a global passive adversary (A.6.1) that might collude with corrupted mix nodes (A.6.2). Finally, we show that our claims also extend to active attacks (A.6.3) as described in the Loopix paper.

A.6.1 Against a global passive adversary

We will show that a global passive adversary (GPA) that can monitor the entire network traffic does not gain any advantage when MultiSphinx messages (A.5.2) are used compared to regular Sphinx messages (A.5.1).

Indistinguishability of MultiSphinx and Sphinx messages for GPA

We first show that a global passive adversary (GPA) cannot tell apart MultiSphinx messages and regular Sphinx messages. By doing this we reduce our security claims to those of the original Loopix paper. We treat all encryption and pseudo-random functions (PRFs) as random oracles as it is done in the original Sphinx paper.

Lemma 29. *MultiSphinx messages from the sender to the multiplication node are indistinguishable from Sphinx messages for a GPA.*

Proof. The headers of all MultiSphinx messages from the sender to the multiplication node are constructed using the same methods as regular Sphinx messages. At the same time the Sphinx header is "[...] hiding the number of hops a messages has travelled so far, as well as the actual number of mixes on the path of a message" [44, p.2]. Therefore, the headers are indistinguishable between the two message types. To an adversary who does not know the key the encrypted payload is indistinguishable from random bits for both message types as per definition of the random oracle. Therefore, the encrypted payloads are indistinguishable between the two message types. □

Lemma 30. *MultiSphinx messages from the multiplication node to other nodes are indistinguishable from Sphinx messages for a GPA.*

Proof. The proof for the header bytes follows analogously to the proof above. The payloads of MultiSphinx messages leaving the multiplication node consist of the encrypted payload concatenated with the pseudo-random padding (the output ρ_i of the PRF). As per the definition of the random oracle, both bit strings are indistinguishable from random noise. Therefore, the entire payload is indistinguishable. This is also true for all following hops, as all messages (before the recipient unpacks the innermost message) are ciphertexts in the random oracle model. □

Lemma 31. *All MultiSphinx messages are indistinguishable from Sphinx messages for a GPA.*

Proof. Since Loopix is using a stratified topology, all message paths will go through a mix node at the multiplication layer. This means that each message (and edge) of the path is either before or after a multiplication node. Lemma 29 and Lemma 30 cover both cases. □

Unlinkability of Messages at Multiplication Node

Theorem 1 of in the Loopix paper [110] analyses the probability that an adversary can link a single message leaving a mix node to one of the previously arriving messages. Our argument follows the same structure to show MultiSphinx maintains the unlinkability property described in the Loopix paper. The MultiSphinx protocol does not affect mix nodes in the first and third layers since they retain a one-to-one relation between incoming and outgoing messages. Therefore the analysis in the Loopix paper holds unchanged. We diverge from the original notation by using κ instead of k to avoid confusion with the k parameter used for our schedule generation.

Theorem 1 in the Loopix paper defines the observation scenario $o_{n,\kappa,l}$ for a passive adversary: first, the attacker observes a set of n messages arriving at a previously empty mix (multiplying into pn messages internally); then a total of $(pn - \kappa)$ messages are emitted by the mix before another set of l messages arrive at the mix; finally, a single message m leaves the mix node and the adversary seeks to correlate this message m with any of the $n + l$ messages observed arriving at the mix node.

Lemma 32. *Let m_1 be any of the initial n messages arriving at the p-restricted MultiSphinx multiplication mix node in scenario $o_{n,\kappa,l}$. Let m_2 be any of the l messages that arrive later. The probability that the outgoing message m was an inner message of either m_1 or m_2 is:*

$$Pr(m \in m_1) = \frac{\kappa}{n(\kappa + pl)}, \tag{A.1}$$

$$Pr(m \in m_2) = \frac{p}{\kappa + pl}. \tag{A.2}$$

Proof. With the arrival of n messages and their multiplication the mix node holds pn messages. After emitting $(pn - \kappa)$ messages the mix holds κ messages. The arrival of the l messages leads to a total of $\kappa + pl$ messages from which m is chosen.

The probability that m is an inner message of any of the initial n messages is $\frac{\kappa}{\kappa+pl}$. The requirement that it was an inner message of one particular message m_1 of that batch which leads to: $\frac{1}{n} \cdot \frac{\kappa}{\kappa+pl} = \frac{\kappa}{n(\kappa+pl)}$.

The probability that m is an inner messages of any of the later l messages is $\frac{pl}{\kappa+pl}$. The requirement that it was an inner message of one particular message m_2 of that batch which leads to: $\frac{1}{l} \cdot \frac{pl}{\kappa+pl} = \frac{p}{\kappa+pl}$. □

These results are qualitatively the same as in the Loopix paper: "[...] continuous observation of a Poisson mix leaks no additional information other than the number of messages present in the mix" [110, p.1207]. As the original paper's argument builds upon the probabilities in its Theorem 1. It also holds for MultiSphinx.

Theorem 33. *A global passive adversary (GPA) that monitors all network traffic does not gain any advantage when MultiSphinx messages are used instead of Sphinx messages.*

Proof. An observer cannot distinguish any MultiSphinx message from a regular Sphinx message (Lemma 31). The difference in processing at the multiplication node also does not provide any advantage (Lemma 32). Furthermore, the multiplication factor p is globally fixed and therefore cannot leak any information about group sizes. All other mix nodes operate exactly as they do with normal

Sphinx messages. To an observer all clients create indistinguishable packets at a rate independent of actual communication. Therefore, an observer does not gain any advantage over regular Sphinx messages. □

A.6.2 Against corrupt nodes

We now consider the ability of an adversary who controls a subset of mix nodes along the message path. The adversary can inspect the internal state of the corrupted mix node including short-term and long-term secrets. We exclude active attacks for now, as they are covered in A.6.3. This is sometimes called an honest-but-curious mix node model.

Theorem 34. *An adversary controlling an honest-but-curious mix node on the first or third layer processing MultiSphinx messages does not gain any advantage compared to regular Sphinx messages for any of the security notions discussed in the original Loopix paper: sender-recipient third-party unobservability, sender online unobservability, sender anonymity, receiver unobservability, and receiver anonymity.*

Proof. From Theorem 33 it follows that the adversary does not gain an advantage from monitoring the in-coming and out-going messages. Also, Algorithm 16 shows the operation of these nodes is identical to regular Sphinx nodes. □

Theorem 35. *An adversary controlling an honest-but-curious multiplication mix node on the second layer processing MultiSphinx messages does not gain any advantage compared to regular Sphinx messages for any of the security notions discussed in the original Loopix paper: sender-recipient third-party unobservability, sender online unobservability, sender anonymity, receiver unobservability, and receiver anonymity.*

Proof. A p-restricted MultiSphinx multiplication node on the second layer does not learn more information about the sender or recipient than a regular Sphinx node on the second layer: it knows its preceding node on the first layer and it knows for each message the succeeding node in the third layer. The addition of deterministic pseudo-random padding $\rho_{\{A,B,\dots\}}$ does not reveal any information about the sender or recipient as it is derived from a shared secret that is known in the regular Sphinx operation as well. □

Definition 36. *In an anonymity system having **group existence anonymity** an adversary cannot decide whether a group of a given size is communicating.*

This property does not hold for naïve multicast: a multiplication node observing a message splitting into x other messages can deduce an increased likelihood that a group of that size exists.

Theorem 37. *An adversary controlling an honest-but-curious mix node processing p-restricted MultiSphinx cannot decide group existence with probability better than random chance.*

Proof. All MultiSphinx messages are constructed independently of the underlying Rollercoaster algorithm (Figure 4.3) and always have the same number of p wrapped messages. None of the observable properties change if there is an actively communicating group of certain size or not. □

Definition 38. *In an anonymity system having* ***group membership anonymity*** *an adversary cannot determine whether two users are members of the same group or not.*

Theorem 39. *Assume a system using 3 mix layers and p-restricted multicast at the middle layer. Assume an adversary, who controls an honest-but-curious multiplication node, and also controls c out of n mix nodes in the third layer. Let $\mathcal{U}$ be the set of all users evenly distributed among all providers $\mathcal{P}$. Then this adversary cannot decide group membership anonymity with probability better than $(1 - (\frac{n-c}{n})^{p-1}) \cdot \frac{|\mathcal{P}|^2}{|\mathcal{U}|^2}$ for a given group message.*

Proof. We assume (to the advantage of the adversary) that MultiSphinx messages contain either payloads to members of the same group or cover traffic. In the real system MultiSphinx messages might contain a mixture of both making the adversary's job more difficult. We also ignore (to the advantage of the adversary) any Loop messages injected by the mix nodes.

We first analyse the chance of the adversary controlling at least 2 of the independently chosen p mix nodes from the third mix layer, as they need both to link the messages.

$$\begin{aligned}
Pr(X \geq 2) &= 1 - Pr(X = 0) - Pr(X = 1) \\
&= 1 - \left(\frac{n-c}{n}\right)^p - p \cdot \frac{c}{n} \cdot \left(\frac{n-c}{n}\right)^{p-1} \\
&= 1 - \left(\frac{n-c}{n} + \frac{p \cdot c}{n}\right) \cdot \left(\frac{n-c}{n}\right)^{p-1} \\
&= 1 - \frac{n + (p-1)c}{n} \cdot \left(\frac{n-c}{n}\right)^{p-1} \\
&\leq 1 - \left(\frac{n-c}{n}\right)^{p-1}
\end{aligned}$$

Because the adversary controls both a multiplication node and multiple layer-3 nodes, they can trace messages that resulted from the same multiplication up to their delivery to the recipients' providers P_1 and P_2. If the traced messages contained payload of the same group, the adversary now knows that there exist users U_1 and U_2 that are likely to be members of the same group and their respective providers are P_1 and P_2. However, without also controlling the providers, the adversary does not know the identity of U_1 and U_2.

The likelihood of correctly guessing the actual recipient for a given provider is $Pr_{provider \to user} = \frac{|\mathcal{P}|}{|\mathcal{U}|}$.

Therefore, the likelihood that an attacker, who controls a multiplication node, correctly deduces two recipients of a message that belong to the same group is:

$$\begin{aligned}
Pr_{win} &= Pr(X \geq 2) \cdot (Pr_{provider \to user})^2 \\
&\leq (1 - \left(\frac{n-c}{n}\right)^{p-1}) \cdot \frac{|\mathcal{P}|^2}{|\mathcal{U}|^2}
\end{aligned}$$

□

The following example applies Theorem 39 to a specific scenario: We assume there are $|\mathcal{U}| = 1000$ users evenly distributed among $|\mathcal{P}| = 10$ providers using p-restricted MultiSphinx with $p = 2$. We

assume that the adversary controls $c_3/n_3 = 20\%$ of multiplication nodes in the third layer. We also assume that the adversary controls $c_2/n_2 = 20\%$ of all multiplication nodes in the second layer, i.e. Theorem 11 applies to only 20% of all messages. These numbers mean that for each group message sent through the network the adversary has a chance of $4 \cdot 10^{-6}$ to correctly name two users who are members of that group. For an adversary controlling $\frac{c_2}{n_2} = \frac{c_3}{n_3} = 50\%$, the probability increases to $2.5 \cdot 10^{-5}$. We leave more precise statistical analysis (leading to a lower upper boundary) for future work.

A.6.3 Against a global active adversary

We now consider a global active adversary (GAA) that can do everything the observer from the previous sections can do. In addition the GAA can also inject/drop network messages and participate as a limited number of users. These abilities match those provided in the Loopix paper.

Lemma 40. *MultiSphinx messages are resistant to tagging attacks.*

Proof. The improved Sphinx construction protects the integrity of the entire message [20]. This includes the deterministic pseudo-random padding ρ_i in our construction. Therefore, modifications to the messages are detected by the mix nodes and such messages will not be processed. □

Theorem 41. *A global active adversary (GAA) that monitors all network traffic and can modify messages does not gain any advantage when MultiSphinx messages are used instead of Sphinx messages.*

Proof. Mix nodes of all levels use the same protections against active attacks as those in the Loopix paper – namely: integrity check of messages against adversarial tagging, loop traffic against n-1 attacks, and message tags against replay attacks. Lemma 40 shows that all MultiSphinx messages are resistant against tagging attacks as well. Therefore, MultiSphinx provides no advantage to an GAA compared to normal Sphinx messages. □

A.7 Reproduced latency distributions

To check the general soundness of our simulation we reproduced the latency distribution provided by the Loopix paper [110, Figure 11]. For their experimental setup with $\lambda_\mu = 2$ the original authors suggest fitting a Gamma distribution with mean 1.93 and standard deviation 0.87. This translates into a shape-scale-parameterised Gamma distribution with parameters $\Gamma_{original}(k \approx 4.95, \theta \approx 0.39)$.

When we compare our data ($n \geq 18000$ measurements) against $\Gamma_{original}$ the fit is not perfect (Figure A.5). The discrepancy can be explained by the fact that the original paper's $\Gamma_{original}$ was determined experimentally and is affected by imprecision caused by their measurement and implementation. However, we can analytically determine the distribution of the sum of the four independent exponential distributions $exp(\lambda_\mu)$: one for the ingress provider and three for the mix nodes. This distribution is $\Gamma_{theory}(k = 4, 0.5)$ and the data from our simulation fits it very well (Figure A.6).

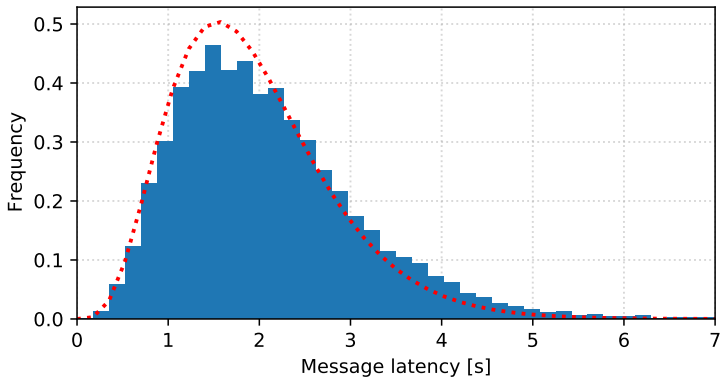

Figure A.5: Distribution of latency measured by our simulator and the Gamma distribution $\Gamma_{orginial}$ (dotted red line) from the original Loopix paper.

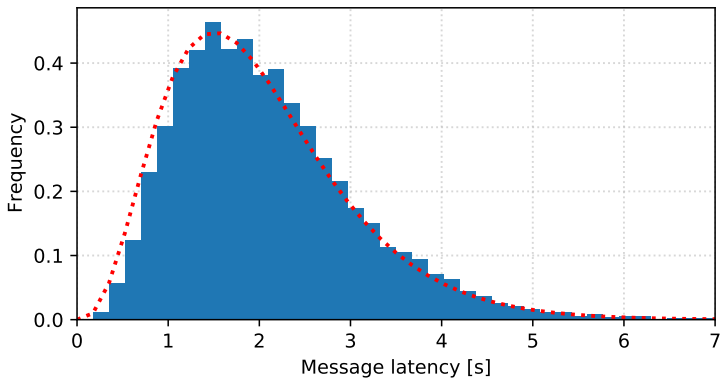

Figure A.6: Distribution of latency measured by our simulator and the Gamma distribution Γ_{theory} (dotted red line) determined analytically.

A.8 Visualisation of offline models

The following figures show samples of 20 nodes and their online/offline schedule. When a node is online it is marked with a blue dot. The percentage provided next to the Y-axis shows the total fraction of the 24h time span that a node is online.

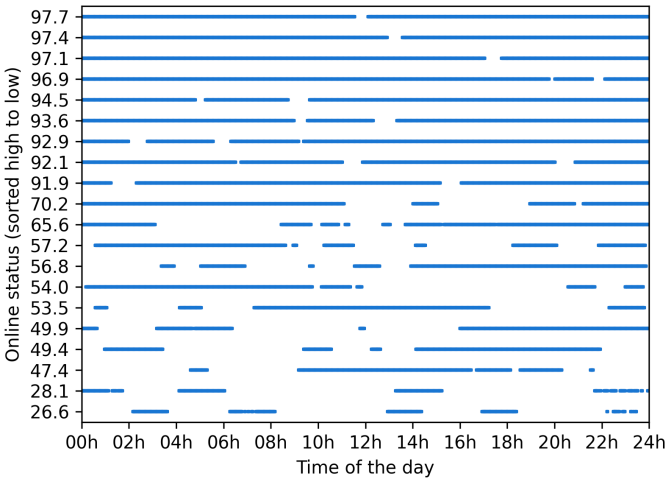

Figure A.7: Sample of 20 nodes for the original model with an average online ratio of 65.05%.

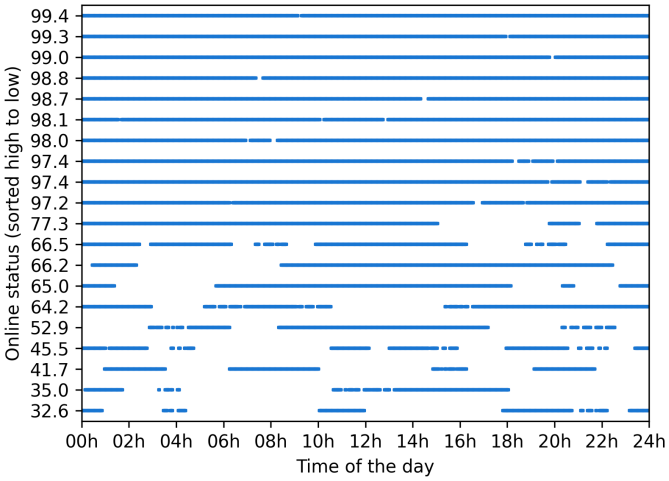

Figure A.8: Sample of 20 nodes for the extrapolated model with an average online ratio of 80.01%.

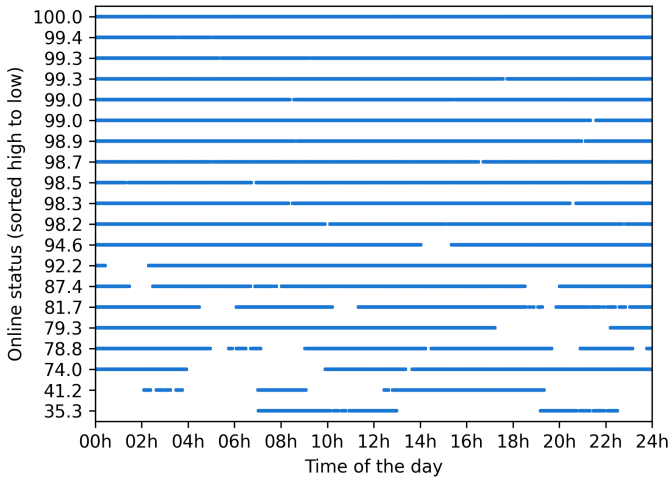

Figure A.9: Sample of 20 nodes for the extrapolated model with an average online ratio of 88.45%.

Appendix B

Sloth

B.1 Security proofs for LongSloth

B.1.1 LongSloth Indistinguishability

Proof. The LongSloth protocol Ξ runs on a SE with HMAC support SE-with-Hmac (Definition 4); let SE-OP be the HMAC protocol run by the SE. We prove Theorem 10 by reduction. Let's assume there exists an efficient adversary $\mathcal{A}_\Xi$ against Ξ; we build an adversary $\mathcal{A}$ against SE-OP. $\mathcal{A}$ interacts with a SE-OP-challenger $\mathcal{C}$ and simulates a Ξ-challenger to $\mathcal{A}_\Xi$.

Definition 42 recalls the IND-HMAC$_\mathcal{A}(\lambda)$ experiment played by $\mathcal{A}$ and $\mathcal{C}$. Bellare proved that HMAC is a PRF under the sole assumption that its underlying compression function is a PRF [22]. As in the original paper by Bellare, it is convenient to consider a PRF-adversary $\mathcal{A}$ that takes inputs (Section 3.2 of [22]). Algorithm B.1 illustrates how to leverage $\mathcal{A}_\Xi$ to build an efficient adversary $\mathcal{A}$ breaking the IND-HMAC security of SE-OP (Definition 43).

Definition 42 (IND-HMAC Experiment [22])**.** *Let λ be a fixed security parameter. Let $\mathcal{A}$ be a WT adversary with wall time budget B and $\mathcal{C}$ a WT challenger. The SE-OP indistinguishability experiment IND-HMAC$_\mathcal{A}(\lambda)$ is defined as follows:*

1. *Let the state ψ be freshly initialised and h an arbitrary (but fixed) key handle. $\mathcal{C}$ randomly samples a secret key $k \leftarrow$ SE.HmacKeyGen(ψ, h) and sets $\psi.h \leftarrow k$.*
2. *$\mathcal{A}$ receives oracle access SE.Hmac under the WT conditions (Definition 3).*
3. *$\mathcal{A}$ submits two chosen plaintexts $(msg_0, msg_1) \in M^2$ to $\mathcal{C}$.*
4. *$\mathcal{C}$ randomly samples $b \xleftarrow{\$} \{0,1\}$, and provides $\mathcal{A}$ with $c_0 \leftarrow$ SE.Hmac(ψ, h, msg) if $b = 0$ or $c_1 \xleftarrow{\$} \{0,1\}^\lambda$ otherwise.*
5. *$\mathcal{A}$ outputs a bit b' and wins iff $b = b'$.*
6. *The experiment returns 1 iff $\mathcal{A}$ wins, otherwise 0.*

Definition 43 (IND-HMAC Security [22])**.** *A SE-OP protocol is IND-HMAC secure if for all WT adversaries $\mathcal{A}$ with time budget B, there is a function negl such that for all λ,*

$$Pr[\text{IND-HMAC}_\mathcal{A}(\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda)$$

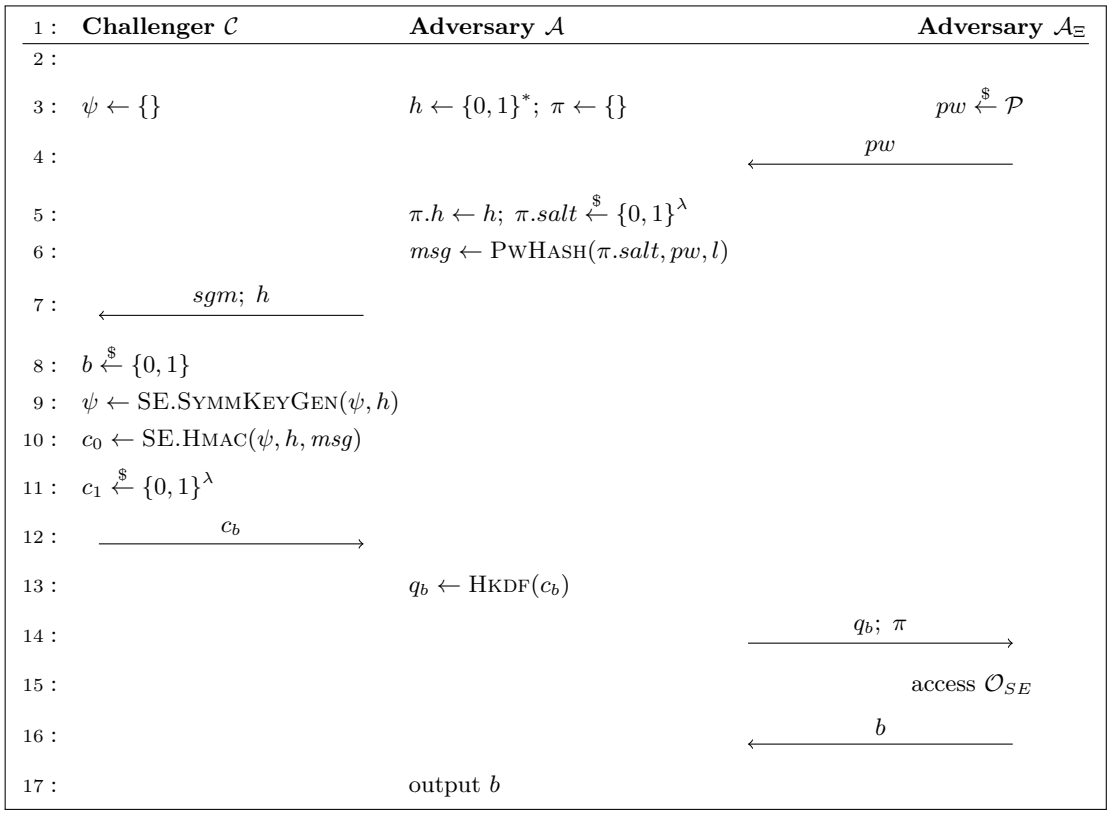

Figure B.1: LongSloth security reduction. $\mathcal{A}$ leverages the efficient adversary $\mathcal{A}_\Xi$ to play against $\mathcal{C}$ and break the IND-HMAC security of SE-OP.

The challenger $\mathcal{C}$ starts by initialising its internal state ψ with a secret key and the adversary $\mathcal{A}_\Xi$ selects a m-entropy secure password pw. $\mathcal{A}_\Xi$ sends pw to $\mathcal{A}$ who uses it to generate a message for the challenger $\mathcal{C}$; it sets $msg \leftarrow \text{PwHash}(salt, pw, l)$ (where $salt$ is a random salt). Challenger $\mathcal{C}$ samples a random bit. If $b = 0$ it provides $\mathcal{A}$ with the HMAC of m, i.e. $c_0 \leftarrow \text{Hmac}(\psi, h, msg)$; otherwise it samples a random bit string $c_1 \xleftarrow{\$} \{0,1\}^\lambda$ of the same size as the HMAC output. In order to guess whether c_b is the output of a HMAC or a random bit string, the adversary $\mathcal{A}$ provides $\mathcal{A}_\Xi$ with the stretched key $q_b \leftarrow \text{Hkdf}(c_b)$. $\mathcal{A}_\Xi$ determines whether q_b originated from its password pw or a random source. To this purpose, $\mathcal{A}_\Xi$ can access the oracle $\mathcal{O}_{SE}$ under the WT conditions (Definition 3). It finally returns $b = 0$ if it believes q_b originated from pw and $b = 1$ otherwise. Finally, the adversary $\mathcal{A}$ deduces that $\mathcal{C}$ computed the HMAC of m if $b = 0$ and that it sampled a random bit string if $b = 1$.

We observe that $\mathcal{A}$ wins the IND-HMAC$_\mathcal{A}(\lambda)$ experiment with the same probability as $\mathcal{A}_\Xi$ wins KeyInd$_{\mathcal{A}_\Xi}(\lambda, \mathcal{P})$. As a result, the existence of an efficient adversary $\mathcal{A}_\Xi$ winning KeyInd$_{\mathcal{A}_\Xi}$ with probability $p > 1/2 + \text{NEGL}$ implies the existence of an efficient adversary $\mathcal{A}$ winning IND-HMAC$_\mathcal{A}(\lambda)$ with the same probability p. This directly violates the assumption that SE-OP runs a IND-HMAC secure HMAC algorithm, hence a contradiction. □

B.1.2 LongSloth Hardness

Proof. We constructively proof Theorem 14 by deriving the success rate of $\mathcal{A}$ for a given wall time budget. For this we first note that k only allows $\mathcal{A}$ to verify its guesses, but provides no helpful information otherwise. This is because the pre-image resistance of HKDF implies $\mathcal{A}$ does not learn any information about ω_{post} (and thus any of the previous state) from k. Hence, k provides $\mathcal{A}$ with no advantage when choosing pw' candidates.

Second, we show that $\mathcal{A}$ has to consider each pw' candidate independently. For this we observe that the output $\omega_{pre} = \text{PWHASH}(\pi.salt, pw')$ is indistinguishable from random as per our assumptions, i.e. the input of SE.HMAC is independent for each pw. We note that the salt sampled as per $\Xi.\text{KEYGEN}$ rules out any pre-computations by $\mathcal{A}$.

Third, we show that $\mathcal{A}$ has to pay the full costs $(l * c_{\text{HMAC}})$ for each of their guesses. This follows from the *existential unforgeability under adaptive chosen-messages* [76, p.113] that we can assume for SE.HMAC. In particular, $\mathcal{A}$ does not gain any information from submitting a prefix of ω_{pre}. With the pre-image resistance of HKDF, this implies that $\mathcal{A}$ must perform all steps of the $\Xi.\text{DERIVE}(\psi, pw', h)$ using oracle $\mathcal{O}_{SE}$. Hence, one password guess reduces $(l \cdot c_{\text{HMAC}})$ units from the adversary's budget B.

Fourth, we derive the probability of success for a set password guesses. Let X be a random variable drawn from the distribution $\mathcal{P}$ (having min-entropy m) by the challenger to determine the password pw. Assume to the advantage of $\mathcal{A}$ that they can efficiently sample a set G of passwords from $\mathcal{P}$ with each $g_i \in G$ independently drawn from $\mathcal{P}$ as random variable Y with $g_i \neq g_j \forall i, j$. Then the probability that one of the guesses allows $\mathcal{A}$ to derive k is: $Pr[\mathcal{A}\ wins] = \sum_{pw \in \mathcal{P}} \sum_{g \in \mathcal{G}} Pr[X = pw] \cdot Pr[Y = pw'] = \sum_{pw \in \mathcal{P}} Pr[X = pw] \cdot \sum_{g \in \mathcal{G}} Pr[Y = pw'] = |G| \cdot Pr[Y = pw'] \leq \frac{|G|}{2^m}$ Based on our arguments above, the maximum size of G that the adversary can verify given budget B is $|G| = \frac{B}{(l \cdot c_{\text{HMAC}})}$. Substituting in the previous equation yields: $Pr[\text{KEYHARD}_{A,\Xi}(\lambda, \mathcal{P}) = 1] \leq \frac{B}{(l \cdot c_{\text{HMAC}})} \cdot \frac{1}{2^m}$. □

B.2 Security proofs for RainbowSloth

B.2.1 RainbowSloth Indistinguishability

Proof. The RAINBOWSLOTH protocol Ξ runs on a SE with symmetric encryption support SE-WITH-ECDH (Definition 5); let SE-OP be the DDH secure Diffie-Hellman key exchange run by the SE. We prove Theorem 11 by reduction. Let's assume there exists an efficient adversary $\mathcal{A}_\Xi$ against Ξ; we build an adversary $\mathcal{A}$ against SE-OP. $\mathcal{A}$ interacts with a SE-OP-challenger $\mathcal{C}$ and simulates a Ξ-challenger to $\mathcal{A}_\Xi$.

Definition 44 recalls the (generalised) decisional Diffie-Hellman (DDH) [17] experiment played by $\mathcal{A}$ and $\mathcal{C}$. Algorithm B.2 illustrates how to leverage $\mathcal{A}_\Xi$ to build an adversary $\mathcal{A}$ breaking the DDH assumption (Definition 45).

Definition 44 (Generalised DDH [17])**.** *Let λ be a fixed security parameter. Let n be an integer and $\mathbb{G}$ a large cyclic group of prime order q. Let $\mathcal{A}$ be a WT adversary with wall time budget B and $\mathcal{C}$ a WT challenger. The generalised DDH indistinguishability experiment $DDH_{\mathcal{A}}(\lambda)$ is defined as follows:*

1. *$\mathcal{A}$ provides $\mathcal{C}$ with $(g_1, \ldots, g_n) \in \mathbb{G}^n$.*

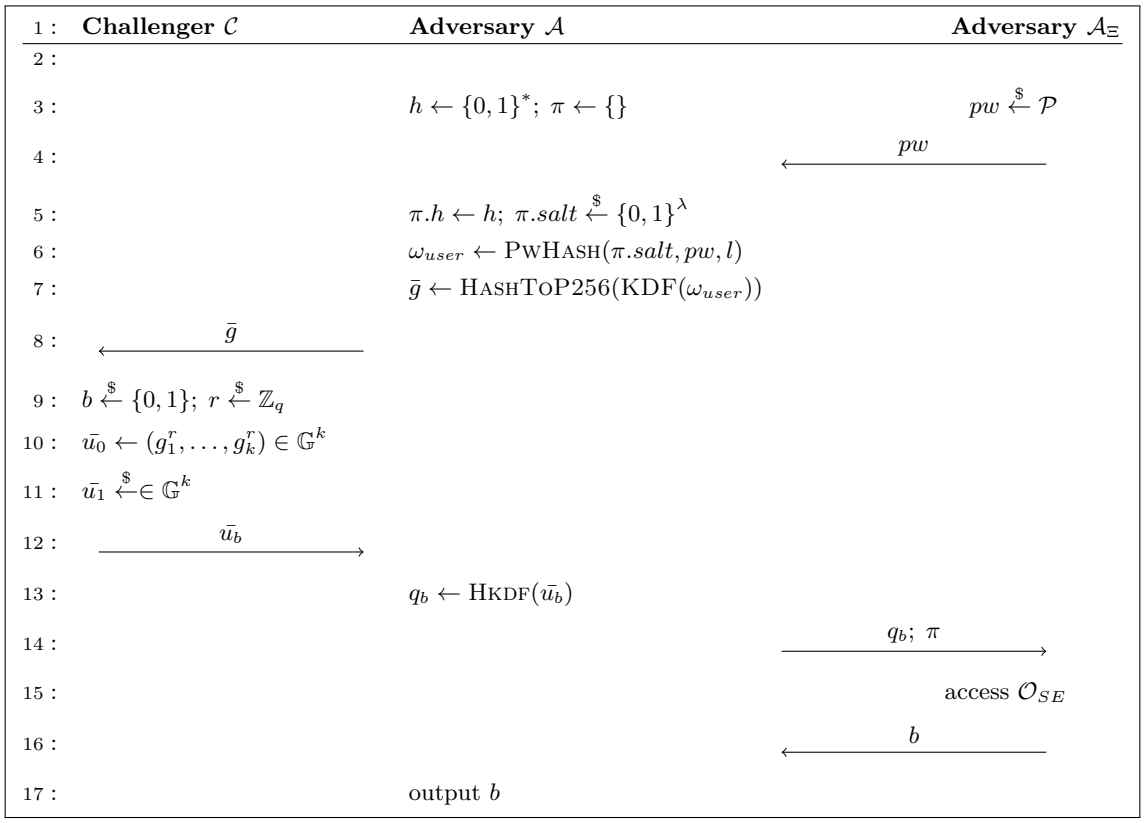

Figure B.2: RainbowSloth security reduction. $\mathcal{A}$ leverages the efficient adversary $\mathcal{A}_\Xi$ to play against $\mathcal{C}$ and break the generalised DDH assumption.

2. *$\mathcal{C}$ randomly samples $b \xleftarrow{\$} \{0,1\}$. If $b = 0$ it randomly samples $r \xleftarrow{\$} \mathbb{Z}_q$ and sets $\bar{u_0} = (g_1^r, \ldots, g_n^r) \in \mathbb{G}^n$; otherwise it randomly samples $\bar{u_1} = (u_1, \ldots, u_n) \in \mathbb{G}^n$. It then provides $\mathcal{A}$ with $\bar{u_b}$.*
3. *$\mathcal{A}$ outputs a bit b' and wins iff $b = b'$.*
4. *The experiment returns 1 iff $\mathcal{A}$ wins, otherwise 0.*

Definition 45 (Generalised DDH [17])**.** *The generalised decisional Diffie-Hellman assumption holds in $\mathbb{G}$ if for any n, there is a function NEGL such that,*

$$Pr[DDH_{\mathcal{A}} = 1] \leq \frac{1}{2} + \textit{NEGL}(\lambda)$$

Algorithm B.2 shows how to leverage the RainbowSloth adversary $\mathcal{A}_\Xi$ to build an efficient adversary breaking the the generalised DDH assumption (Definition 45).

□

B.2.2 RainbowSloth Hardness

Proof. The proof for RainbowSloth is analogous to the one for LongSloth (Section B.1.2) with the cost of the critical operation exchanged for $n \cdot c_{\text{ECDH}}$ as per Definition 5. □

B.3 Security proofs for HiddenSloth

Both 1S-HiddenSloth and MS-HiddenSloth run on top of a key stretching scheme such as LongSloth (Section 5.2.3) and RainbowSloth (Section 5.2.4); as a result, they run on a SE with either HMAC or ECDH support. As mentioned in Section 6 and Section 7, any HiddenSloth protocol Δ requires an authenticated IND-CPA secure stream cipher AE running in the user space (i.e. outside of the SE).

B.3.1 MS-HiddenSloth Indistinguishability

Proof. We prove Theorem 19 by reduction. Let's assume there exists an efficient adversary $\mathcal{A}_\Delta$ against Δ; we build an adversary $\mathcal{A}$ against AE. $\mathcal{A}$ interacts with a AE-challenger $\mathcal{C}$ and simulates a Δ-challenger to $\mathcal{A}_\Delta$. Definition 46 recalls the IND-CPA experiment played by $\mathcal{A}$ and $\mathcal{C}$; Definition 47 recalls the definition of AE's IND-CPA security.

Definition 46 (AE IND-CPA Experiment)**.** *Let λ be a fixed security parameter. Let $\mathcal{A}$ be a WT adversary with wall time budget B_λ and $\mathcal{C}$ a WT challenger. The AE indistinguishability experiment $IND\text{-}CPA_{\mathcal{A}}(\lambda)$ is defined as follows:*

1. *$\mathcal{C}$ randomly samples a secret key $k \leftarrow AE.KeyGen$.*
2. *$\mathcal{A}$ receives oracle access $AE.Enc$ under the WT conditions (Definition 3).*
3. *$\mathcal{A}$ submits two chosen plaintexts $(msg_0, msg_1) \in M^2$ to $\mathcal{C}$.*
4. *$\mathcal{C}$ randomly samples $b \xleftarrow{\$} \{0,1\}$ and $iv \xleftarrow{\$} IV$, and provides $\mathcal{A}$ with $c_b, t_b \leftarrow AE.Enc(k, iv, msg_b)$.*
5. *$\mathcal{A}$ receives oracle access $AE.Enc$ under the WT conditions.*
6. *$\mathcal{A}$ outputs a bit b' and wins iff $b = b'$.*
7. *The experiment returns 1 iff $\mathcal{A}$ wins, otherwise 0.*

Definition 47 (AE IND-CPA)**.** *A stream cipher AE is IND-CPA secure if for all WT adversaries $\mathcal{A}$ with time budget B, there is a function negl such that for all λ,*

$$Pr[IND\text{-}CPA_{\mathcal{A}}(\lambda) = 1] \leq \frac{1}{2} + negl(\lambda)$$

It is convenient to grant $\mathcal{A}$ access to a SE oracle allowing to perfectly simulate SE symmetric encryption operations $SE.SymmEnc$ as performed by a SE with symmetric encryption support SE-with-SymmEnc (Definition 6). This oracle is described in Algorithm 17 and is initialised calling O.Init before starting the experiment.

Algorithm 17 Oracle simulating $SE.SymmEnc$ operations as performed by a SE-with-SymmEnc.

```
1: procedure O.Init()
2:     ψ ← {}
3: procedure O.Encrypt(h', msg)
4:     ψ ← SE.SymmKeyGen(ψ, h')
5:     iv ←$ IV
6:     returnSE.SymmEnc(ψ, h', iv, msg)
```

Algorithm B.3 illustrates how to leverage the MS-HiddenSloth adversary $\mathcal{A}_\Delta$ to build an efficient adversary $\mathcal{A}$ breaking the IND-CPA security of AE (Definition 47). Algorithm B.3 generates a random

key k calling AE.KeyGen. The experiment should ideally generate k through a Sloth key stretching algorithm but Theorems 10 and 11 state that $\mathcal{A}_\Delta$ cannot distinguish a key generate by a Sloth algorithm from a purely random key.

During the preparation phase (lines 7 to 21 of Algorithm B.3), the adversary $\mathcal{A}_\Delta$ provides $\mathcal{A}$ with a message $\tilde{msg} \leftarrow M$ and a bit $\tilde{b}$. $\mathcal{A}$ makes use of it oracle access to AE.Enc (step 2 of Definition 46) by forwarding $\tilde{msg}$ to the challenger $\mathcal{C}$. $\mathcal{C}$ encrypts $\tilde{msg}$ and replies with the corresponding iv iv, ciphertext $\tilde{c}$, and tag $\tilde{t}$. $\mathcal{A}$ stores the queried message as $s_{\tilde{b}} \leftarrow \tilde{msg}$; it will later use it to simulate the state of the protocol Δ to $\mathcal{A}_\Delta$. It then sample a fresh key tk to re-encrypt $\tilde{c}$, which simulates the outer encryption layer of Algorithm 7. $\mathcal{A}$ makes use of its SE encryption oracle (defined in Algorithm 17) to encrypt tk, $\tilde{t}$, and iv. $\mathcal{A}$ now holds all information to craft a state π as if generated by a Δ-challenger. The preparation phase repeats a number of times chosen by $\mathcal{A}_\Delta$ (under WT conditions).

$\mathcal{A}$ eventually prepares two messages for the challenger $\mathcal{C}$: $msg_0 \leftarrow s_0$ and $msg_1 \leftarrow s_1$. The values s_0 and s_1 retain the latest messages queried by $\mathcal{A}_\Delta$ during the preparation phase for $\tilde{b} = 0$ and $\tilde{b} = 1$, respectively. The challenger $\mathcal{C}$ encrypts either msg_0 or msg_1 based on a random bit b and provides $\mathcal{A}$ with the corresponding ciphertext and tag $c_b, t_b \leftarrow \text{AE.Enc}(k, iv, msg_b)$ and iv iv. At this point, c_b is the encryption of the latest pair $(\tilde{msg}, b)$ queried by $\mathcal{A}_\Delta$ during the preparation phase (at Line 7). $\mathcal{A}$ finally re-encrypt those information to simulate the outer encryption layer similarly to the preparation phase.

$\mathcal{A}_\Delta$ accesses the oracle $\mathcal{O}_{SE}$ and eventually determines whether π contains the encryption of the latest pair $(\tilde{msg}_0, 0)$ or $(\tilde{msg}_1, 1)$ it submitted during the preparation phase. It eventually returns $b = 0$ if it believes π contains the encryption of $\tilde{msg}_0$ and $b = 1$ otherwise. Finally, the adversary $\mathcal{A}$ deduces that $\mathcal{C}$ encrypted msg_0 if $b = 0$ and msg_1 if $b = 1$.

We observe that $\mathcal{A}$ wins the IND-CPA$_\mathcal{A}(\lambda)$ experiment with the same probability as $\mathcal{A}_\Delta$ wins experiment DE-MS-Ind$_{\mathcal{A}_\Delta}(\lambda, \mathcal{P})$. As a result, the existence of an efficient adversary $\mathcal{A}_\Delta$ winning experiment DE-MS-Ind$_{\mathcal{A}_\Delta}(\lambda, \mathcal{P})$ with probability $p > 1/2 + \text{negl}$ implies the existence of an efficient adversary $\mathcal{A}$ winning IND-CPA$_\mathcal{A}(\lambda)$ with the same probability p. This directly violates the assumption that AE is IND-CPA secure, hence a contradiction. □

B.3.2 MS-HiddenSloth Hardness

Proof. We prove Theorem 22 by showing that the probability that adversary $\mathcal{A}$ wins the deniable encryption hardness experiment for protocol Δ is no bigger than its probability of winning the hardness experiment against its underlying key stretching scheme Ξ. That is, $Pr[\text{KeyHard}_{\mathcal{A},\Delta}(\lambda, \mathcal{P}) = 1] \leq Pr[\text{KeyHard}_{\mathcal{A},\Xi}(\lambda, \mathcal{P}) = 1]$.

First, $\mathcal{A}$ removes the outer encryption layer calling SE.SymmDec and AE.Dec (line 25 and line 26 of Algorithm 7). As a result, $\pi.blob$ contains the encryption of $data$, encrypted using a key derived from pw. This operations requires a one-time cost of $\text{T}_{\text{SE.SymmEnc}}$ from the adversary's budget B and does not need to be repeated for each guess.

Second, we observe that π (and $\pi.blob$ in particular) only allows $\mathcal{A}$ to verify their guesses, but provides no helpful information otherwise. This observation follows from the IND-CPA resistance of the underlying AE encryption scheme. Hence, knowing π provides $\mathcal{A}$ with no advantage when choosing pw' candidates.

Third, we show that $\mathcal{A}$ has to pay the full cost c_Ξ required to access the SE and run the underling key stretching scheme Ξ used by Δ. DECRYPT. $\mathcal{A}$ must find a key k such that AE.DEC$(k, \pi.iv, \pi.blob, \pi.tag)$ returns *data* (line 16 of Algorithm 6). Assuming AE is a permutation-based cipher, there is a single k satisfying this property. Let's assume (to the advantage of the adversary) that $\mathcal{A}$ can brute-force AE.DEC to recover k (since it does not require access to the SE and thus does not cost $\mathcal{A}$'s budget). $\mathcal{A}$ must now win the key stretching hardness experiment presented in Definition 12 against a Ξ-challenger. Assuming Ξ is σ_Ξ-hard, Definition 13 indicates $\mathcal{A}$ pays budget $c_\Xi = \sigma_\Xi$ for each of their guesses.

Finally, since the adversary pays a one-time cost $\mathrm{T}_{\text{SE.SYMMENC}}$ and a cost of σ_Ξ for each of their guesses, the overall cost of guessing pw is $\sigma > \sigma_\Xi$ per guess. It follows that $\frac{B}{\sigma \cdot 2^m} < \frac{B}{\sigma_\Xi \cdot 2^m}$, and thus $Pr[\text{DEHARD}_{\mathcal{A},\Delta}(\lambda, \mathcal{P}) = 1] \leq Pr[\text{KEYHARD}_{\mathcal{A},\Xi}(\lambda, \mathcal{P}) = 1]$. □

	Challenger $\mathcal{C}$	**Adversary $\mathcal{A}$**	**Adversary $\mathcal{A}_\Delta$**
1 :			
2 :	$k \leftarrow \text{AE.KeyGen}()$	$s_0 \leftarrow 0;\ s_1 \leftarrow 0$	
3 :		$h' \leftarrow \{0,1\}^*\|\|1$	
4 :			
5 :			$\tilde{m} \leftarrow M;\ \tilde{b} \leftarrow \{0,1\}$
6 :			$\tilde{m};\ \tilde{b} \longleftarrow$
7 :		$s_{\tilde{b}} \leftarrow \tilde{m}$	
8 :	$\longleftarrow \tilde{m}$		
9 :	$iv \xleftarrow{\$} IV$		
10 :	$\tilde{c}, \tilde{t} \leftarrow \text{AE.Enc}(k, iv, \tilde{m})$		
11 :	$\tilde{c};\ \tilde{t};\ iv \longrightarrow$		
12 :		$tk \leftarrow \text{AE.KeyGen}()$	
13 :		$\pi \leftarrow \{\};\ tiv \xleftarrow{\$} IV$	
14 :		$\pi.blob, \pi.ttag \leftarrow \text{AE.Enc}(tk, tiv, \tilde{c})$	
15 :		$\pi.tiv \leftarrow tiv;\ \mathcal{K} \leftarrow [tk, t_b, iv]$	
16 :		$\pi.\mathcal{K} \leftarrow \text{O.Encrypt}(h', \mathcal{K})$	
17 :			$\pi \longrightarrow$
18 :			
19 :		$m_0 \leftarrow s_0;\ m_1 \leftarrow s_1$	
20 :	$\longleftarrow m_0; m_1$		
21 :	$b \xleftarrow{\$} \{0,1\};\ iv \xleftarrow{\$} IV$		
22 :	$c_b, t_b \leftarrow \text{AE.Enc}(k, iv, m_b)$		
23 :	$c_b;\ t_b;\ iv \longrightarrow$		
24 :		$tk \leftarrow \text{AE.KeyGen}()$	
25 :		$\pi \leftarrow \{\};\ tiv \xleftarrow{\$} IV$	
26 :		$\pi.blob, \pi.ttag \leftarrow AE.Enc(tk, tiv, c_b)$	
27 :		$\pi.tiv \leftarrow tiv;\ \mathcal{K} \leftarrow [tk, t_b, iv]$	
28 :		$\pi.\mathcal{K} \leftarrow \text{O.Encrypt}(h', \mathcal{K})$	
29 :			$\pi \longrightarrow$
30 :			access $\mathcal{O}_{SE}$
31 :			$b \longleftarrow$
32 :		output b	

Figure B.3: MS-HiddenSloth security reduction. $\mathcal{A}$ leverages the efficient adversary $\mathcal{A}_\Delta$ to play against $\mathcal{C}$ and break the IND-CPA security of AE.