Private User Discovery in Anonymous Communication Networks

Ceren Kocaoğullar

King's College

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science

University of Cambridge
Department of Computer Science and Technology
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
UNITED KINGDOM

Email: ck596@cam.ac.uk

June 11, 2021

Declaration

I Ceren Kocaoğullar of King's College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed: Ceren Kocaoğullar

[illegible]

Date: 11 June 2021

Abstract

Private communications have recently become an integral part of our everyday lives with encrypted messaging applications such as Signal, Telegram, and WhatsApp. On the other hand, PGP, a twenty-year-old encryption system mostly used for email, has historically seen limited adoption. A likely explanation for this disparity is that finding friends on messaging apps is easy via using phone numbers, whereas it is obscure and confusing on PGP, as demonstrated by research [1].

Despite having a practical user discovery mechanism and protecting message contents, encrypted messaging apps do not provide *metadata privacy*. Therefore, they potentially expose critical information about conversations and jeopardise user privacy. Anonymous communication networks can solve this problem. However, similar to PGP, the current user discovery mechanisms for these networks are unusable.

This dissertation identifies the need for a new privacy-preserving and practical user discovery mechanism in anonymous communication networks. To satisfy this need, I have systematically defined the properties of an ideal user discovery mechanism. An analysis of these properties has revealed that the ideal user discovery system is not practically achievable. Therefore, I have established two security protocols, each representing a different point in the usability-privacy tradeoff space: *ID-Verified Pudding* allows user discovery through validated email addresses, but it cannot hide usernames from the user discovery mechanism. *Incognito Pudding* solves this issue at the cost of sacrificing the ability to link Pudding usernames to well-known external names. To investigate the completeness and liveness of these protocols, I have developed a model checker in Python and simulated all possible protocol runs in a finite state space. The results of this evaluation demonstrate that both Pudding protocols are practically achievable.

Word count: 14,915

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Private communications have recently become an essential part of our daily digital conversations with encrypted messaging apps such as Signal [2], Telegram [3], and WhatsApp [4]. In fact, the search for private communication services and many of the cryptographic ideas have been existing for much longer than the lifetime of these applications. One reason behind the newfound widespread adoption of these messaging apps is that they made finding friends practical simply by using phone numbers. A notable earlier example, PGP, an encryption system mostly used for email, fails to enable novice users to find other users practically, as demonstrated by research [1]. The lack of a usable discovery mechanism is likely an important factor in PGP's limited adoption despite its two decades of existence.

Although the mentioned popular messaging apps hide the conversation contents using encryption, they do not achieve strong anonymity as they do not protect *metadata*. Metadata refers to information about a message other than what is said in it, such as when, with whom, and how frequently parties communicate. This type of data can reveal enough information to render encrypting the substance of the messages ineffective. As Former Director of NSA and CIA General Michael Hayden has famously stated, they "kill based on metadata" [5].

Anonymous communication networks aim to provide *metadata privacy* for achieving better privacy than merely encrypting message contents. Tor [6] is perhaps the most well-known anonymous communication network. Nevertheless, it is not the only one, nor does it provide perfect anonymity. Tor achieves low latency and high throughput, which enables real-time communications. However, it does not provide metadata privacy against a global passive adversary, who is capable of observing multiple network links throughout the Internet [7].

On the other hand, mix networks, another type of anonymous communication network, provide metadata privacy against global passive and active adversaries. The higher latency and lower throughput of mix networks make them unfit for real-time applications such as videoconferencing. However, these networks are still able to support asynchronous messaging. As a result, some mix-based anonymous communication networks, such as Loopix [8], Vuvuzela [9], Talek [10], and Pung [11], specialise in messaging with metadata privacy. Every user in these *anonymous messaging systems* possesses some *contact information* that makes them reachable. To be able to contact a user on an anonymous messaging system, one has to learn her contact information, or *discover* them.

Even though the aforementioned anonymous messaging systems provide improved anonymity with metadata privacy, they lack a usable discovery mechanism. This is a critical shortcoming because as illustrated by the PGP versus messaging apps example above, usable discovery is key for enabling communication technologies to be accessible to a broad user base.

A usable discovery mechanism in anonymous messaging systems should allow users to find each other by their human-memorable username. This is analogous to the idea behind DNS and address books: Users easily recall URLs and people's names, not IP addresses or phone numbers, because memorising random-looking strings requires repetition over an extended period, as shown by previous research [12].

In addition to being usable, a discovery mechanism in an anonymous messaging system should provide privacy. Specifically, it is critical for the mechanism to (1) keep discovery relationships between users secret, and (2) make contact information reachable only to those who know the username. These two requirements separate user discovery in anonymous messaging systems from the rest of the discovery space. Although mentioned above as usable lookup mechanisms, neither

DNS nor address books meet these requirements. In fact, there is seemingly no existing solution that meets the usability and privacy demands in this problem area. This study aims to delve into this critical yet underexplored issue and develop novel approaches and solutions to make anonymous messaging systems like Loopix more usable while preserving the metadata privacy promises of anonymous communication networks.

Regarding the second privacy requirement, all discovery mechanisms are subject to the friction between privacy and usability: Memorable usernames are often also guessable [13]. This may allow adversaries to determine registered usernames and obtain contact information by simply querying the discovery mechanism. Notwithstanding this inherent limitation, in this dissertation, I will propose ways to protect usernames and contact information from being easily accessed in large quantities.

This dissertation makes the following contributions:

- Identifying the need for a new usable and private user discovery mechanism in anonymous communication networks (Section 3.1).
- A novel framework describing the requirements of an ideal privacy-preserving and usable user discovery system from operational, usability, and security & privacy perspectives (Section 3.3).
- An extensive analysis of system goals, as well as threat and trust models. This analysis reveals that an ideal user discovery mechanism is unattainable, as some system goals cannot coexist (Sections 3.2 and 3.4).
- Two original user discovery protocols meeting different subsets of the ideal system's goals, and might apply to different settings (Chapters 4 and 5).
- First-time application of the Oblivious Pseudorandom Function (OPRF) concept to the user discovery space.
- A simulation tool to demonstrate practicality, completeness and liveness of the protocols. This tool assesses the protocols with a model checking approach, mechanising the processes and simulating all protocol runs within a finite state space (Chapter 6).

Chapter 2

Background

This chapter describes anonymous communication networks and reviews the current literature about private discovery in a broad context. It begins by describing anonymous communication networks and identifies that one type of these networks, anonymous messaging systems, has special requirements for discovering users. Then, this chapter presents the prevalent mechanisms used for discovering peers in relevant areas. However, no existing mechanism meets the criteria for being used as a user discovery mechanism in anonymous messaging systems. Finally, this chapter provides the foundation for some key cryptographic concepts that are used for designing protocols in consecutive chapters.

2.1 Anonymous Communication Networks

Anonymous communication networks aim to allow users to share information with others without revealing who they are communicating with to observers of network traffic. These networks do more than merely encrypting the transmitted data; they aim to provide metadata privacy, hiding information such as which users communicate, when, and how often.

As mentioned in the previous chapter, although Tor [6] has brought anonymous communication networks to the mainstream, this line of research has produced a myriad of designs that cover many different areas of use and employ various technologies. Tor anonymises TCP-based communications like browsing the Web, peer-to-peer real-time communications, and secure shell [14]. Other application areas for anonymous communication networks include anonymous emails [15], e-voting [16, 17, 18], anonymous credentials [19, 20], anonymous Voice-over-IP calls [21], file sharing [22], electronic currencies [23], anonymous bulletin boards [24], and anonymous auctions [25].

These different areas of specialisation come with different requirements. For instance, networks aiming to anonymise credentials should allow users to prove their identity without revealing it. On the other hand, those that are used in e-voting have to be exceptionally robust. Therefore, there is no generic formula to designing an anonymous communication network. Different architectures employ different technologies such as mixing [26], onion routing [27], and Dining Cryptographers networks (DC-nets) [28] with different approaches.

This dissertation focuses on anonymous communication networks that allow anonymous messaging, or *anonymous messaging systems*, such as Loopix [8], Vuvuzela [9], Talek [10], and Pung [11]. Regardless of their technical intricacies, all these systems have a requirement in common: In order to communicate with a user, one has to learn some network-related information about them, in other words, *discover* them. Although the need for discovering network entities is common among many types of anonymous and other communication networks, some specific anonymity requirements separate user discovery from other forms of node discovery. These requirements are outlined in the next section and described in detail in Section 3.1,

2.2 Private Discovery in Other Contexts

This section describes some private discovery mechanisms used in settings other than anonymous communication systems and discusses their suitability to user discovery. Encrypted messaging apps such as Signal [2] aim to allow users to find their contacts registered to the app easily and privately. *Private contact discovery* covers the line of research and industry practices that study this issue. The first three parts of this section discuss the three main ideas used in private contact discovery, namely Private Set Intersection (PSI), Private Information Retrieval (PIR), and using secure enclaves. The last two parts describe DNS and Distributed Hash Tables (DHT), the two major mechanisms used for node discovery in various anonymous communication networks.

There are two main privacy requirements that a user discovery mechanism has to meet to retain the level of privacy that anonymous messaging networks provide. The first one is preventing the discovery mechanism from linking the user with the users she discovers. The second one is hiding the user's contact information from everyone including the discovery nodes, unless they make legal queries to discover the user. None of the mechanisms presented in this section meets both these privacy criteria. Therefore, they cannot be used for user discovery in anonymous messaging systems.

2.2.1 Private Set Intersection (PSI)

The line of research on Private Set Intersection (PSI) aims to devise protocols where parties figure out the intersection of sets of information without learning anything else about each other's sets, except maybe their sizes [29]. PSI is a suitable and widely discussed solution for the mainstream private contact discovery problem, i.e. determining which users in a user's address book are registered in the system without learning nothing about the rest of the user's address book. However, PSI is not directly applicable to user discovery in anonymous communication networks, since this technique does not allow users to privately obtain information that is necessary for messaging others. As the name suggests, PSI merely works for privately finding intersections of sets of data.

2.2.2 Private Information Retrieval (PIR)

Private Information Retrieval (PIR) defines a class of cryptographic techniques that allows retrieval of items from a server without revealing to the server which items are retrieved [11]. PIR techniques can be grouped under two categories: Computational PIR (C-PIR) and Information-Theoretic PIR (IT-PIR).

C-PIR methods support a very strong trust model, placing confidence in no one but the communication partner, at the cost of high computational or network costs. IT-PIR techniques are usually computationally cheaper than C-PIR schemes, however, they support the weaker *anytrust* threat model, i.e. trusting that at least one database is honest [10].

Although PIR could be helpful in achieving user discovery, significant computation and network costs of even the less costly IT-PIR raise questions about this technique's practicality and scalability. Moreover, these schemes are usually designed for database queries using multiple keywords, whereas queries needed for user discovery are of a much simpler form, and therefore suitable for more straightforward solutions.

2.2.3 Secure Remote Computation

Another idea explored in the mainstream private contact discovery field is using secure remote computation for secretly finding the intersection of the user's address book and the users registered to the messenger service. Specifically, Signal's private contact discovery mechanism [30] employs Intel Software Guard Extensions (SGX) technology, which offers a secure remote computation environment [31]. Combining this feature with the right computational approach, Signal aims to achieve the desired privacy in private contact discovery. However, changing the hardware in existing systems to support this technology is costly and burdensome. Moreover, the SGX technology has been prone to attacks that can extract private data stored in the enclave [32].

2.2.4 Private DNS Solutions

Domain Name Service (DNS) is used for translating human-readable domain names to IP addresses by recursively querying servers based on a determined hierarchy [33]. This mechanism was originally designed to make information publicly available and therefore did not aim to provide any privacy. However, as it started to be used in services requiring stronger privacy, DNS security and privacy grew into a topic of interest in the research community [34]. DNSSEC was introduced by the Internet Engineering Task Force (IETF) for ensuring the integrity and authenticating the origin of DNS data. However, these extensions do not aim to provide privacy [35].

Most of the existing research about DNS privacy focuses on protecting DNS queries and responses from unauthorised parties. To provide a base for private DNS queries and responses, OpenDNS has devised two solutions: DNSCrypt [36] and DNSCurve [37], which both use elliptic-curve cryptography to encrypt DNS communications. Apart from these two best-known methods, several more drafts have been submitted to enter IETF's standards track, such as Confidential DNS [38] and Private DNS [39]. In addition to these commercial and organisational efforts, protecting the privacy of DNS queries and responses has also been a subject of academic research [40, 41, 42, 43].

All these DNS solutions aim to hide the contents of DNS queries and responses. However, they do not aim to prevent linking users with their queries. Therefore, none of these private DNS solutions is suitable for user discovery. More recent approaches like Oblivious DNS [44], DNS for Tor [45], and distributed DNS [46] solve this issue. Nevertheless, these solutions do not meet the other privacy requirement for user discovery, which is hiding contact information from the discovery mechanism.

2.2.5 Distributed Hash Tables (DHTs)

The second family of mechanisms that are used for node discovery in various types of anonymous communication networks is Distributed Hash Tables (DHTs). As the name suggests, a DHT is a distributed system that functions like a hash table, which allows peers in the system to search for data objects using the keys associated with them.

As with DNS, most DHT implementations do not provide protection against linking users with queries. The ones that aim to provide this property, including Salsa [47], AP3 [48], NISAN [49], Torsk [50] are susceptible to information leakage, as illustrated by numerous studies [49, 51, 52, 53].

2.3 Cryptographic Background

This section aims to lay the groundwork for some key cryptographic concepts that are essential for building the user discovery mechanisms presented in the following chapters of this dissertation.

2.3.1 Anonymous Replies

Anonymous replies refer to a specific message type used in anonymous communication networks, which allows users to send a message to a receiver and receive a response from them while remaining anonymous [15, 54]. A primitive called *single-use reply block* (SURB) [15] is used for enabling anonymous replies.

One way for Alice to create a SURB is to construct a mix header with an empty payload destined for *herself*. This resulting header includes a path consisting of a number of relay nodes and finally arriving at Alice. Due to the nature of mix headers, this header follows a path that goes to Alice, while not revealing her identity to anyone but the last relay node in the path. Therefore, when Alice wants to send an anonymous message to Bob and receive a reply without revealing her identity, she attaches a SURB to her message. The SURB allows Bob to reply to Alice without knowing who she is. Ultimately, Alice enjoys *sender anonymity* when sending the message, and *receiver anonymity* when receiving a reply [54].

2.3.2 Threshold Secret Sharing

The concept of secret sharing was first proposed in 1979, independently by Shamir using polynomial interpolation [55] and Blakley based on hyperplane geometry [56]. Although numerous secret sharing schemes with different properties have been proposed since then (e.g. [57, 58, 59, 60, 61, 62]), the core idea behind all secret sharing schemes is the same: They allow dividing a secret into pieces in such a way that no piece makes sense by itself, but the pieces can be collectively used to reconstruct the secret.

A threshold secret sharing scheme, or (k, n) *threshold scheme*, is a secret sharing scheme with a special property: The secret is split into n shares and distributed to different participants. *Any* group of k or more participants can assemble their shares to reconstruct the secret, but no group with less than k members can [55].

2.3.3 Hash Function

A hash function is a deterministic function that maps arbitrary-sized bit strings to fixed-sized bit strings. These functions have two useful properties; they are:

1. *one way*, i.e. knowing only the output of a hash function $H(x)$, it is computationally infeasible to find x except by brute-force guessing its value
2. *collusion resistant*, i.e. it is computationally infeasible to find two different inputs $x \neq y$ that yield the same hash output $H(x) = H(y)$ [63]

2.3.4 Oblivious Pseudorandom Function (OPRF)

A Pseudorandom Function (PRF) is a function that takes a random seed and a data value as inputs, and creates an output that is indistinguishable from an actual random value [64].

An Oblivious Pseudorandom Function (OPRF) defines a class of protocols between two parties that allows the parties to securely and collectively compute the keyed PRF $F_k(x) = y$. This dissertation calls these two parties the *key* (k) *holder* and the *input* (x) *provider*. As a result of performing an OPRF, the *input provider* learns the output of the function (y), whereas the *key holder* does not learn anything [65]. To explain the concept better, a case study is presented below.

OPRF Case Study: Password Hardening

OPRF can be used for deriving a strong secret from a weak secret, such as a low-entropy password. One example to this is a *password-to-random* protocol proposed by Jarecki et al. [66] based on Ford-Kaliski password hardening technique [67]. This protocol can be described as below.

Alice wants to use a human-memorable password in an authentication process. She also wants to prevent adversaries from learning her password through online and offline attacks. To achieve this, Alice performs an OPRF with a designated device (D) she can communicate with, such as her mobile phone or a server. In this OPRF, Alice is the *input provider* and the device is the *key holder*. Alice provides her password (pwd) as an input to the OPRF and obtains a randomised value. This is a pseudorandom value which Alice can consistently obtain from her password when she collaborates with D, but is irreversable and unlinkable to her password. Alice can use the randomised value to compute a strong randomised password (rwd), with which she can register and authenticate to an authentication server. Due to the nature of OPRFs, the device D does not learn anything about the password during the process.

Below is the OPRF protocol that this study proposes to achieve the described goal:

- Alice hashes her password using a hash function H' that maps from inputs to elements of a prime-order cyclic group G. Then, she picks a random number ρ and uses this number to *blind* the hash value $H'(pwd)$ by raising the value to the power of ρ. The blinding step hides the value of pwd from anyone who does not know the randomly chosen ρ. Alice sends this blinded hash value to D. ($H'(pwd)^{\rho}$ where $H' : \{0,1\}^* \rightarrow G$).
- Upon receiving this blinded hash of the password, if the value is in the cyclic group G, D raises it to the power of D's secret key, k, and replies to Alice with the resulting value. ($H'(pwd)^{\rho k}$).
- Alice *de-blinds* the received value by raising it to the power of $1/\rho$ to retrieve $H'(pwd)^k$. She then uses a different hash function to hash this value with password. The resulting value is Alice's randomised password rwd. ($rwd \leftarrow H(pwd, H'(pwd)^k)$ where $H : \{0,1\}^* \rightarrow \{0,1\}^{\tau}$)

2.3.5 Digital signatures

A digital signature is a cryptographic primitive, which enables checking the integrity of a message and authenticating the originator. A digital signature scheme can be described as follows: A secret signing key (sk) and a corresponding, but non-identical public verification key (pk) are used for creating a signature and verifying a message-signature pair, respectively. A signed message and its signature are presented together to the party which will validate the signature. Through this scheme, only the owner of the secret signing key can sign a message ($sig \leftarrow Sign(msg, sk)$); however, everyone can check the signature's validity using the public verification key ($SigVerify(pk, msg, sig)$) [68, 69].

2.3.6 DomainKeys Identified Mail (DKIM)

DomainKeys Identified Mail (DKIM) [70] is a method that uses digital signatures to confirm the origin of emails. In other words, by checking the DKIM signature, one can be confident that an email came from the domain it claims that it did.

The basic idea is as follows: The email contents, specifically the hash of the body and some selected header fields, are signed using the signer's secret signing key (sk). This signature, namely *DKIM signature*, is added to a *DKIM header* and attached to the email. The recipient can then retrieve the public verification key (pk) that the signer has published to the DNS as a TXT record and use it to verify the DKIM signature [70, 71].

The DKIM signature is not the only information that the DKIM header contains; many other required and optional fields exist. Figure 2.1.a shows the fields that are included in a sample header. The most significant fields in a DKIM header are as follows:

- `b:` The DKIM signature
- `h:` List of the email's header fields included in the signature
- `bh:` Hash of the email's *canonicalised* body: Canonicalisation aims to transform the message so that different ends of the communication using different encodings calculate the same hash value
- `d:` Domain of the signing entity
- `s:` Selector used for partitioning the namespace of `d`

Although the signer of an email can directly be the sender, as the name suggests, DKIM is originally intended to execute signing and verification at the *domain* level. Therefore, the signer is often an entity higher in the domain hierarchy [70].

(a) Raw DKIM signature:

```
 v=1; a=rsa-sha256; c=relaxed/relaxed;
d=UniversityOfCambridgeCloud.onmicrosoft.com;
s=selector2-UniversityOfCambridgeCloud-onmicrosoft-com;
h=From:Date:Subject:Message-ID:Content-Type:MIME-Version:X-MS-Exchange-
→ SenderADCheck;
bh=GuD+k1F9m43ShzdNbvA0XehinfuLo1kj43NJkwxvg08=;
b=gX/0ACfK4PqBP20PccyptrTD4w5DYYIoTpD4wkD6Gu9YVT3ufEgc1JxJeR/etVrXdDyUM/
→ JDuVlo0t4lSSplvQfaWH6SQcZ3EIjPvDrmV7/U5wuf3qK1/B6DZpm8N4vn69ByWaSk8Vu
→ WU+6dovz0RaQ3jiAj8KyEx0bY0LYNeAk=
```

(b) Signature information explained:

```
 v= Version:  1
a= Algorithm:  rsa-sha256
c= Method:  relaxed/relaxed
d= Domain:  UniversityOfCambridgeCloud.onmicrosoft.com
s= Selector:  selector2-UniversityOfCambridgeCloud-onmicrosoft-com
q= Protocol:
bh= GuD+k1F9m43ShzdNbvA0XehinfuLo1kj43NJkwxvg08=
h= Signed Headers:  From:Date:Subject:Message-ID:Content-Type:MIME-
→ Version:X-MS-Exchange-SenderADCheck
b= Data:  gX/0ACfK4PqBP20PccyptrTD4w5DYYIoTpD4wkD6Gu9YVT3ufEgc1Jx
→ JeR/etVrXdDyUM/JDuVlo0t4lSSplvQfaWH6SQcZ3EIjPvDrmV7/U5wuf3qK1/
→ B6DZpm8N4vn69ByWaSk8VuWU+6dovz0RaQ3jiAj8KyEx0bY0LYNeAk=
```

(c) Public key DNS lookup

```
 Building DNS Query for
selector2-UniversityOfCambridgeCloud-onmicrosoft-com._domainkey.
→ UniversityOfCambridgeCloud.onmicrosoft.com
Retrieved this publickey from DNS: v=DKIM1; k=rsa;
p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC92TmjHQXbQcV1JqUW/IEsjGW8+i+9Q3
→ 9EWJZh9BnPGGkL5bSHoFevHIi6fDN0af3D/ab8a0etbCVslr0Ag8voucAKKfa3MkF1Q58mYC+
→ MwUkI1EPVNHzzpUVLRn2XALrxrx8WrdFuS0PLTWJrptK60sBERE034o4zqxUJvUIsWQIDAQAB;
```

Figure 2.1: (a) DKIM signature of an email sent from my Cambridge email address, (b) explained version of the signature, and (c) details about the DNS lookup for obtaining the public verification key. Data is obtained through `https://dkimvalidator.com`. The arrow signs (→) denote line breaks.

For instance, as Figure 2.1 illustrates, the emails that I send from my Cambridge email address are signed by domain `d=UniversityOfCambridgeCloud.onmicrosoft.com` using selector `s=selector2-UniversityOfCambridgeCloud-onmicrosoft-com`. As described in Figure 2.1.c, verifiers of the emails that I send from this address make a DNS lookup using these two identifiers to retrieve my domain's public verification key. This key is then used for verifying the DKIM signature attached to the email.

2.4 Summary

This chapter has described some key concepts that are necessary for this dissertation. The first of these concepts is anonymous communication networks, i.e. networks that aim to provide strong privacy properties by hiding metadata as well as the communicated data. Among many use areas of these networks, this dissertation focuses on anonymously sending messages to others. This chapter has also presented some mechanisms used for discovery in different contexts, as well as providing the foundation in some cryptographic notions which are necessary to navigate the ideas presented in the remainder of this dissertation.

Chapter 3

Goals and Problem Statement

This chapter describes the problem that this dissertation aims to solve: The need for a usable and privacy-preserving discovery system in anonymous communication networks. It also methodically defines and analyses the objectives of an ideal user discovery mechanism and details its threat model. Recognising that it is not possible to implement an ideal scheme which reaches all the defined system goals, this chapter introduces two implementations which attain different sets of goals and have different tradeoffs.

3.1 Problem Description

Users of anonymous communication networks need to know some information about each other to be able to communicate. The needed information differs among schemes. Often, if not always, it includes a user's public encryption key, which, in some cases, is used as a system identifier as well, e.g. in Talek [10] and Vuvuzela [9]. However, depending on the network's architecture, knowing a user's public key might not always be enough to communicate with them. For instance, Loopix [8] requires users to know each other's network location as well, which consists of the user's network identifier and her provider's IP address. I use the term *contact information* to refer to this information.

Definition 3.1.1. Contact Information: The information that one user has to know to contact another user through an anonymous communication network.

The act of learning a user's contact knowledge is called *user discovery.* This term is defined as:

Definition 3.1.2. User Discovery: A function $f(x) = y$ where x is a human-readable string identifying the user and y is the *contact information* of that user.

User discovery on anonymous communication networks should not be esoteric, but comprehensible by the expected user base. Existing anonymous communication research fails to address the need for user discovery, assuming that users somehow already have the appropriate contact information [8, 9, 10, 11]. However, user discovery is key for preventing anonymous communication networks from becoming a privilege of individuals who understand their technicalities and are willing to make extra effort to adapt to their eccentricities. Poor user discovery is one of the reasons why PGP, an encryption system mainly used for email, is remarkably difficult to use, while encrypted messaging services such as Signal [2], Telegram [3], WhatsApp [4] are wildly popular. The seminal work of Whitten and Tygar has shown that one of the main issues with PGP 5.0 was its explicit key generation/distribution processes and inability to navigate users through them, which made it very difficult for cryptography novices to contact other users using PGP [1]. In other words, this research illustrated that *user discovery* in PGP has *poor usability*, as it is too difficult for everyday users to execute.

Having mentioned the terms *usability* and *privacy*, this dissertation defines them in the context of user discovery as:

Definition 3.1.3. Usability: A user discovery function is usable if (1) users do not have to leave the primary communication system and communicate through an external channel to perform it, and (2) its input can be a human-memorable username, and if desired, an established public identifier such as phone number or email address.

Definition 3.1.4. Privacy: A user discovery function is private if no party other than Alice, who is a user initiating the user discovery for Bob (1) learns Bob's contact information or (2) can link Alice to Bob.

Many practical applications that are concerned with sharing contact information do not fit the usability definition above, as they require users to exchange contact information through external channels (e.g. sharing links, reading QR codes). This is not only a usability problem. More importantly, it undermines the whole purpose of using an anonymous communication network by leaking the information that certain users are communicating through some potentially insecure channel, such as SMS.

On the other hand, usable discovery mechanisms such as DNS and DHT are not *private* according to the above definition, as described in Sections 2.2.4 and 2.2.5. Even private DNS solutions do not provide privacy as defined above, since they do not prevent the DNS servers from linking queries with users, although they satisfy the second condition by encrypting queries. On the other hand, most DHT implementations do not satisfy the second condition, and the ones that aim to satisfy it are vulnerable to information leakage. As a result, there is no existing solution in the user discovery space that is both usable and private.

Another family of usable discovery mechanisms is called *private contact discovery* [29, 72, 73, 74]. Some encrypted messaging apps employ private contact discovery techniques for allowing users to discover others using phone numbers. Since phone numbers are already-known identifiers typically stored in the user's mobile address book, this approach makes the privacy offered by these messaging systems accessible to everyday users. However, existing private contact discovery applications do not achieve the first privacy goal since they do not hide the contact information from servers that store it.

Ultimately, a *usable* and *privacy-preserving* mechanism by which users can discover other users on an anonymous messaging system has not been established. Moreover, a systematic understanding of the requirements of this solution space is still lacking. As previously stated, I propose an ideal user discovery mechanism to define these requirements. Before discussing that in Section 3.3, the next section describes the threat and trust assumptions of this ideal mechanism.

3.2 Threat Model

I assume that my user discovery mechanism runs on top of an anonymous communication network and comprise four types of entities: (1) *Users*, who use the discovery mechanism to discover and be discovered by other users; (2) *User devices*, which allow users to connect to the user discovery mechanism and the underlying anonymous communication network; (3) *Discovery nodes*, which are responsible for storing user data and responding to discovery queries; and (4) *Discovery node operators*, who manage the discovery nodes.

In terms of security, I assume that the user discovery mechanism employs a (k, n) threshold secret sharing scheme for storing contact information in the discovery nodes. Adversaries are assumed to be interested in disrupting the user privacy defined in Definition 3.1.4. In other words, adversaries try to link users to their discovery queries and learn registered user information, i.e. contact information and/or username. It is important to note that since the discovery mechanism runs on an anonymous communication network, the usual network attackers eavesdropping and actively interfering with packets do not pose a threat.

As I pointed out in the introduction to this dissertation, this discovery mechanism cannot hide contact information from an adversary who acts as a user and issues legal discovery queries. Similarly, such an attacker can query the system with bulk data to compile a list of registered usernames. This problem, called *crawling*, is considered an intrinsic issue of discovery systems, as the main function of these systems is allowing users to find others registered to the system [13]. However, this user discovery mechanism provides much stronger security when it comes to hiding who is searching for whom.

Adversarial assumptions of the system can be examined further on the basis of actor types:

1. **Users** can be ***malicious***. They might try to *crawl* usernames and contact information. Users may also try to change other users' contact information.
2. **User devices** can also be ***malicious***. They may be curious, passively collecting information. Also, they may actively deviate from the protocol,

e.g withholding messages from being sent or received or sending arbitrarily malformed messages.

3. **Discovery nodes** are also considered potentially ***malicious***. Similar to user devices, they can perform passive attacks by secretly gathering information, or active attacks by providing faulty answers, using wrong functions, not responding, etc.

4. **Discovery node operators** can be ***malicious under a non-collusion assumption***, i.e. no single operator should control more than or equal to a determined threshold (k) number of discovery nodes.

5. **Attackers** can act as any or multiple of the actors mentioned above; he can control user devices or discovery nodes, act as a user, or be a discovery node operator as long as he controls less than threshold-many discovery nodes.

An attacker with the described capabilities does not compromise security or privacy through active attacks, except the intrinsic *crawling* issue. However, he might degrade the availability of the system. Specifically, the discovery mechanism cannot maintain its functionalities if the number of malfunctioning or offline discovery nodes is more than the threshold (k) value. Also, a misbehaving or inaccessible user device can prevent the user from using the discovery mechanism.

In terms of passive attacks where an adversary stealthily observes communications, the threat model of the underlying anonymous communication network affects the user discovery mechanism's security and privacy promises. For instance, if I deploy my user discovery mechanism in Loopix, which protects against *global passive adversaries*, my mechanism is also resilient against such adversaries.

3.3 System Goals

This section defines the ideal *usable* (Definition 3.1.3) and *private* (Definition 3.1.4) *user discovery mechanism* (Definition 3.1.2) for anonymous communication networks. It aims to provide a framework that is universal and adaptable to different networks and needs.

The goals that I set for the ideal user discovery mechanism span three categories: *Operational Goals* are the essential goals that a system has to meet to be able to function as a user discovery mechanism. *Security and Privacy Goals* and *Usability Goals* allow the system to not only satisfy the minimum *privacy* and *usability* definitions, respectively, but also enhance it with additional properties. Following the taxonomy strategy that Bonneau et al. use for evaluating password alternatives [75], every goal is referred to with an italicised mnemonic title and labeled with a letter and a number for later reference.

An ideal private user discovery system meets the following goals:

A. Operational Goals

O1 ***registration:*** users can sign up to the system with a username and other required contact information to be discoverable by other users

O2 ***discoverability:*** all registered users can be discovered using the mechanism

O3 ***user-discovery:*** registered users can query the system by providing usernames and learning the corresponding contact information

O4 ***availability:*** the system maintains its operability as long as threshold (k) or more discovery nodes are alive and functioning normally

B. Security and Privacy Goals

S1 ***unlinkability:*** an adversary controlling any nodes or observing any communications is not able to learn *who is searching for whom*, provided that the querying user device is not compromised

S2 ***contact-information-privacy:*** no one (including discovery nodes under the non-collusion assumptions described in Section 3.2) other than the user herself or someone who queries the discovery mechanism with the user's username can learn a user's contact information

S3 ***username-anonymity:*** usernames are stored privately in the system in a way that discovery nodes, even if threshold-many or more of them collude, cannot feasibly learn usernames stored in their databases

S4 ***internal-identity-verification:*** it is possible to confirm that a user is the owner of the username and contact information she claims to own

S5 ***external-identity-verification:*** a user's identity within the system can be *verifiably associated* with an external identity, e.g. an email address

S6 ***membership-unobservability:*** it is not possible to learn if a username belongs to a member of the system or not by querying the system

C. Usability Goals

U1 ***all-internal-execution:*** users do not need to use a secondary channel or interact with an auxiliary system for any functionality that the system offers

U2 ***human-memorable-username-selection:*** users are allowed to pick any string as their username provided that it is not already being used by an existing user

U3 ***authorised-user-data-updates:*** users can make authorised modifications to their contact information on the system, including deleting it

U4 ***account-recovery:*** if a user loses access to her account, she can reclaim it

It is important to note that these goals are not of equal importance. However, providing a fixed scale to quantify definitive values for these system goals is not possible. Different anonymous communication networks in different practical settings might value these goals differently. For instance, *membership-unobservability* may be seen as an essential goal in a scenario where the anonymous communication network is known to be used by the LGBTQ+ community in an oppressive state where members of this community are heavily punished. On the other hand, the same goal might be dispensable in a setting where a great portion of users publishes their usernames on their public-facing social media pages. In essence, these goals aim to paint the picture of a complete user discovery mechanism, while allowing room for different tradeoffs.

	S1	S2	S3	S4	S5	S6	U1	U2	U3	U4
S1										
S2										
S3					●	❍				❍
S4										
S5			●				●			
S6			❍							
U1					●					
U2										
U3										
U4			❍							

Table 3.1: Matrix showing the compatibility of goals: ● marks the goals that are not compatible, and ❍ denotes potential incompatibility under certain conditions, compatible goals are unmarked. *Operational Goals* are excluded, because they are the system's essential goals, and thus are compatible with all other goals.

3.4 Analysis of System Goals

The many system goals described in Section 3.3 describe a conceptual framework. However, these goals might be in conflict in practice. Table 3.1 shows which system goals are **compatible**, i.e. can coexist in an implementation; which are **incompatible**; and which are **potentially incompatible**, depending on implementation choices.

Since *Operational Goals* pertain to the system's the core functionalities, they are compatible with all other goals. Therefore, these are not included in the matrix. Some other goals as well, namely *unlinkability* (S1), *contact-information-privacy* (S2), *internal-identity-verification* (S4), *human-memorable-username-selection* (U2), and *authorised-user-data-updates* (U3), work with every other goal.

For the remaining goals, the reasons causing incompatible or potentially incompatible goal couples are explained as below:

S3 – S5: *username-anonymity* and *external-identity-verification* are **incompatible**, because, the first one hides the usernames from the discovery nodes, making them impossible to be used for communication through external channels. For example, the discovery nodes cannot send an email to someone without knowing their email address or send them a text message without knowing their phone number.

S3 – U4: *username-anonymity* and *account-recovery* are **potentially incompatible** depending on how account recovery is implemented. Specifically, the implementation might practice *account-recovery* by sending a link or code to the user via email, SMS, etc. As *username-anonymity* withholds usernames from the discovery nodes while *account-recovery* requires them to know the user's email address or phone number, the two goals will be in conflict in this scenario.

S3 – S6: *username-anonymity* and *membership-unobservability* are **potentially incompatible**. The only visible way for the discovery system to achieve *membership-unobservability* is to respond to the queries for non-registered usernames with realistic fake contact information. If the system achieves *username-anonymity* in such a way that the discovery nodes cannot link different discovery queries for the same user to each other, then the discovery nodes will not able to answer with the consistent fake contact information for the same non-registered username, which disrupts *membership-unobservability*.

S5 – U1: *external-identity-verification* and *all-internal-execution* are **incompatible**, since verifying ownership of an identifier which can be used for contacting a user through external channels can only by achieved through users communicating with the discovery nodes through that external channel.

3.5 Pudding Protocols

The previous section has illustrated that a single user discovery mechanism meeting all the system goals is unattainable. Therefore, this dissertation introduces a new family of protocols named *Pudding*, which is a play on the abbreviation of *Private User Discovery (PUD)*. I present two concrete Pudding protocols, which satisfy different subsets of these goals and that might be preferable in different contexts. The two protocols are called *ID-Verified Pudding* (Chapter 4) and *Incognito Pudding* (Chapter 5).

ID-Verified Pudding, as the name suggests, verifies user identity by linking user presence within the system with email addresses, achieving *external-identity-verific ation* **(S5)**. However, this design falls short in providing *username-anonymity* **(S3)**, as email addresses are not hidden from the system.

Incognito Pudding solves this privacy issue by hashing usernames with specially generated salt values and storing the resulting hashes in the system instead of plaintext usernames. This solution achieves *username-anonymity* global passive adversaries, however, it sacrifices the *external-identity-verification* **(S5)** feature.

Aside from these differences in username management, both ID-Verified Pudding and Incognito Pudding share a number of security and privacy features. First, they both attain *unlinkability* goal **(S1)** by using anonymous replies (Section 2.3.1) for *all communications between discovery nodes and users.* Second, both systems employ (k, n) threshold secret sharing to meet the *contact-information-privacy* goal **(S2)**. Third, both Pudding protocols allow *internal-identity-verification* **(S4)**, as users can look up each other in the system to confirm their Pudding identities.

3.6 Notation & Terms

This section describes some terms that will be used in the remainder of this dissertation when describing the two Pudding protocols, and provides their notation.

ID: **Username** is the unique identifier that users use to search and discover others through Pudding. It can be an email address, or a system-specific identifier, similar to a social media handle. Preferably, it should be easy to say, read, and remember.

γ: **Contact information** is the information that is necessary for contacting a user via the underlying anonymous communication network (see Definition 3.1.1). Contact information can include different types of information depending on a network's architecture. For instance, in Loopix, this is the user's network address, consisting of her network identifier and the IP address of her provider [8]. Moreover, contact information can include the user's public key (a) if an additional key exchange protocol is undesirable, (b) if it is also used as the user's network identifier as in Vuvuzela [9], or (c) if it is used for bootstrapping as in Pung [11].

D**: Discovery Nodes** hold the information and interact with users to allow them to privately discover other users. These nodes are known to each other and the users, i.e. their network locations, identifiers, and public keys are provided to users when they join the network. Every discovery node has a user registry, which can be thought of as a dictionary. Each user entry in a registry is a $(key, value)$ pair. According to the system design, a key can be an ID or another type of identifier. Every $value$ is user data consisting of a γ secret share and a digital signature public verification key used for signing and updating user data.

A **Searcher** is a user who looks up another user through Pudding.

A **Searchee** is a user who is looked up through Pudding.

3.7 Summary

This chapter has discussed the lack of a user discovery mechanism that makes finding other users on anonymous communication networks as practical as encrypted messaging apps, while protecting user privacy to a greater degree. It also described the threat and trust assumptions, adversaries, and their capabilities in this problem space. An ideal user discovery mechanism which solves this problem and provides additional functionalities is characterised through a number of operational, security & privacy, and usability goals. The chapter acknowledges that this ideal system is practically unattainable. Therefore, two protocols that partially meet these goals, namely ID-Verified Pudding and Incognito Pudding, are described in the next two chapters.

Chapter 4

ID-Verified Pudding

This chapter describes ID-Verified Pudding, one of the two Pudding protocols. By linking user identities within Pudding to email addresses, ID-Verified Pudding allows users to practically search for others and know that the user they discover is actually who she claims to be, assuming at least threshold-many discovery nodes are honest. This chapter begins by laying out the protocol's features, concerns, security and privacy goals, and core idea, which is *identity verification*. It then describes the protocol's functionalities, namely registration, user discovery, updating user data, and account recovery. Finally, it finishes by explaining the shortcomings of the protocol.

4.1 Protocol Overview

ID-Verified Pudding allows users to use their email addresses as usernames. This has two benefits: First, email addresses are non-secret, unique, and mostly easy to remember. Therefore, users can search for others practically, using identifiers that are typically already in their contact address books. Second, using email addresses as usernames enables verifying them through DKIM, allowing users to be confident that someone whom they have discovered through Pudding with an email address is actually in control of that email address.

This *external-identity-verification* feature comes with a privacy tradeoff: It cannot be achieved at the same time with *username-anonymity* (see Section 3.4). This makes the system suitable for those *who want to hide what their contact information is, and whom they are searching for, but are not worried about publicly testifying whether they are a member of the network.* On the other hand, the lack of *username-anonymity* might upset users who require a higher level of anonymity over better usability.

4.1.1 System Goals Analysis

ID-Verified Pudding meets all *Operational Goals*, as well as all *Security and Privacy* but two, namely *username-anonymity* **(S3)** and *membership-unobservability* **(S6)**. This protocol's name comes from its core feature, which is *external-identity-verification* **(S5)**. In addition, the protocol achieves *unlinkability* **(S1)**, since messages from user device to discovery node are sent via the anonymity network, and the messages from discovery node to user device are anonymous replies to the former. Employing a (k, n) threshold secret sharing scheme where contact information is split into secret shares and distributed to discovery nodes, this protocol provides *contact-information-privacy* **(S2)**. Since users are able to use the discovery mechanism to look up the users who initiate communication with them and verify their contact information, ID-Verified Pudding supports *internal-identity-verification* **(S4)** as well.

Regarding *Usability Goals*, ID-Verified Pudding meets three out of four of them. Specifically, this protocol allows users to pick their email addresses as their username, reaching the *human-memorable-username-selection* goal **(U2)**. Moreover, it employs digital signatures to support *authorised-user-data-updates* **(U3)**. ID-Verified Pudding uses the verified external communication channel it establishes through *external-identity-verification* to achieve *account-recovery* goal **(U4)**. On the other hand, since users have to interact with another system to verify their external identity, *all-internal-execution* **(U1)** is not supported.

4.1.2 The Core Idea: Identity Verification

As pointed out in Section 4.1, this Pudding design uses DKIM signatures to achieve *external-identity-verification* **(S5)**, i.e. to verify the ownership of an email address a user picked as a username and to provide trust to those who discover the user.

A single discovery node verifying the DKIM signature of a registration email is not sufficient to verify an email address. Since ID-Verified Pudding employs a (k, n) threshold secret sharing scheme, for the discovery system to confidently say that an email address belongs to a user, at least threshold-many discovery nodes have to verify a user's external identity through DKIM.

It is important to note that although DKIM verifies the source of an email, it cannot verify if the email is genuinely sent for registration. Therefore, at least threshold-many discovery nodes also have to check this. Two straightforward solutions to this issue are: (1) allowing discovery nodes to observe each others' email traffic, or (2) having a system where each discovery node sending an email to the user can satisfy this need. The first solution causes an access control issue, since misbehaving nodes' access to the email address(es) should be revoked. The second solution is problematic in usability terms since it requires users to reply to multiple emails. This usability issue might be circumvented by sending the user a single email where at least threshold-many discovery nodes' email addresses are secondary recipients. However, this solution is prone to user errors. Consequently, neither of these solutions is suitable to verifying email addresses. My system implements a different method to make sure that all discovery nodes involved in registration are also included in the preparation of a single verification email. This method is described below and is also illustrated in Figure 4.1.

Summary of the Authentication Mechanism: When a user initiates registration with an email address, the user assigns one discovery node the responsibility of sending the verification email. All the other discovery nodes send this selected node a randomised challenge value. The selected node also generates a random challenge and sends an email including all these challenge values to the user's email address. The registering user responds with an email that contains challenge values in its body. Since email applications usually attach the original message to the response, the user typically does not have to type or copy-paste any value. Upon receiving the user's response, the selected discovery node distributes the response email's contents and DKIM signature to the other discovery nodes. This allows each discovery node to confirm that the received email is (1) *a genuine registration request* by checking if their challenge value was included in the email and (2) *sent by the owner of the email address* by checking the validity of its DKIM signature.

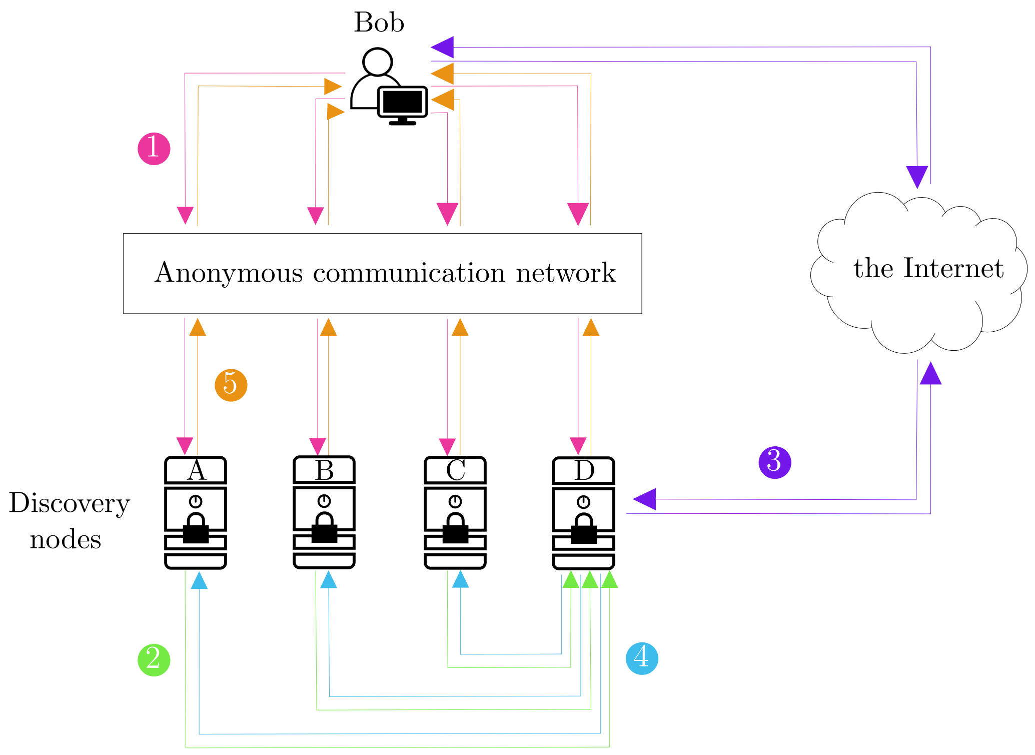
Usability: Although this mechanism disrupts the *all-internal-execution* goal, it is highly practical as it requires the user to typically click only two buttons – *reply* and *send*. Therefore, although responding to an email is not a commonplace account verification practice, its usability is arguably comparable to the prevalent method of clicking a link.

Substitutability of DKIM: Although this implementation uses DKIM, it is not the only viable verification method. If DKIM verification fails, the system can fall back to a method where each discovery node sends the user a verification link. Indeed, this method has the usability downside of requiring multiple confirmation emails and link clicks.

4.2 Registration to the System

The registration phase of the protocol involves important steps for achieving privacy and verification (Figure 4.1). Specifically, to accomplish the *contact-information-privacy* **(S2)** goal, the user *secret shares* her contact information (γ) with the discovery nodes. For achieving the *external-identity-verification* **(S5)**, the discovery nodes verify email usernames (IDs) through DKIM, as explained in Section 4.1.2. Finally, creating and sharing digital signature keys allows the user to make *authorised-user-data-updates* **(S3)** later if needed (Section 4.4).

Below is a step-by-step description of the registration process described in three parts: (A) *Preparation*, (B) *Verification*, and (C) *Registration and Confirmation.* Moreover, the process is described in Algorithm 1 and Algorithm 2, and Figure 4.1.



1 • Bob sends ID, svk, $secret_i$ and D_{auth} (picked as D) to A, B, C, and D

2 • A, B, and C generate random challenges and send them to D

3 • D combines all the challenges in an email and sends this email to Bob

• Bob responds to this email

• D verifies the response email's DKIM signature

4 • D shares the email contents and the DKIM signature with A, B and C

• A, B, and C check that their challenges are included in the email and verify the DKIM signature

5 • A, B, C and D verify DKIM signature and register Bob as $ID : (svk, secret_i)$

• Each discovery node notifies Bob that he was successfully registered

Figure 4.1: Bob's registration process to ID-Verified Pudding.

A. Preparation

1. The registering user chooses an email address that she has access to as her ID **(U2)**.

2. The user's device creates a public-private key pair (pk, sk) for secure and

authenticated communication on the underlying anonymous communication network, and an account update key pair (pk_u, sk_u) to be used for digitally signing data updates later (Section 4.4) **(S3)**.

3. Using a threshold secret sharing scheme such as Shamir's secret sharing [55], the user divides her contact information (γ) into n secret shares **(S3)**.

4. The user randomly picks a discovery node as D_{auth}, i.e. the discovery node responsible for sending the verification email.

5. The user prepares n *registration messages* in the form that allows anonymous replies **(S1)**. The message for each discovery node D_i contains one secret share, the user's ID and pk, as well as D_{auth}'s identifier: $registration\_msg_i \leftarrow (D_{auth}, secret_i, ID, pk, pk_u)$.

6. After sending out the prepared messages to each of the n discovery nodes, the user starts waiting for a *verification email* to arrive at the email address chosen as ID.

   (a) If the user receives a *verification email* within a predetermined timeout period, the registration process continues without disruption.

   (b) Otherwise, the user picks a different D_{auth} and re-sends the registration message, i.e. goes back to Step A.4.

B. Verification **(S5)**

1. After receiving a *registration message*, each discovery node checks if it has a user record with the ID in this message. If ID belongs to an existing record, the *registration message* is dropped and verification at this node fails. The node replies to the user with a suitable error code.

2. Every discovery node also checks if it is selected as D_{auth}.

   (a) If this is the case, it generates a challenge value, and starts waiting for all other discovery nodes to send challenge values.

   (b) Otherwise, it creates a random value as a challenge and sends it to D_{auth}.

3. D_{auth} waits for a predetermined timeout period to collect at least $k - 1$ challenges **(O4)**.

   (a) If D_{auth} receives at least $k - 1$ challenges within this time period, it creates and sends a *verification email* to the email address provided as the user's ID. This email includes all the challenges.

   (b) Otherwise, it drops the *registration message*, aborting the registration and sending the user a relevant error code.

4. The user waits for this *verification email.*

   (a) If the user receives the email within a predetermined timeout period, she responds to it. The only requirement about the response email's content is that it should include all challenge values. Typically, email applications automatically attach the replied email's body to the reply.

   (b) Otherwise, she goes back to Step A.4 to pick a new D_{auth} and try registering again.

5. Upon receiving the user's response, D_{auth} verifies the DKIM signature.

   (a) If the DKIM signature is invalid, D_{auth} drops ID's registration information. User verification and registration fails.

   (b) Otherwise, D_{auth} marks ID as verified. It then sends $n - 1$ encrypted *verification check messages* to all other discovery nodes. Each of these messages include the contents and DKIM signature of the user's response to the *verification email.*

6. Upon receiving a *verification check message* from D_{auth}, each of the $n - 1$ discovery nodes checks (1) if its challenge was included in the email, and (2) if the DKIM signature is valid.

   (a) If both of these conditions hold, it labels ID as verified.

   (b) Otherwise, it drops ID's registration information. Authentication at this node fails.

C. Registration and Confirmation

1. At each discovery node D_i, if verification for ID is successful, the user is registered to the node, i.e. a record for the user is created and added to the user registry as $ID : (secret_i, ID, pk, pk_u)$.
2. Upon registering the user, each discovery node uses the SURB (see Section 2.3.1) attached to the *registration message* to create a *registration confirmation message* destined to the registering user. This message confirms to the user that the registration at this discovery node has been completed.
3. If the user receives at least k *registration confirmation messages* within a predetermined timeout period, the user knows that she is successfully registered and verified in Pudding. For better usability, the confirmation messages do not have to be separately reported to the user but tracked automatically by the user device instead.

ID-Verified Pudding is a distributed system, which can experience network latency and node failures. Therefore, it is important that the mechanism described above meets the *availability* goal: The user can register to the system as long as at least k discovery nodes are functioning properly. If some discovery nodes are unavailable during registration, the user device automatically and periodically retries registering to these nodes.

Algorithm 1: Registering to ID-Verified Pudding as a user

```
user_id ← Pick an email address you have access to
pk, sk ← Generate a public-private key pair for encryption-decryption
$pk_u, sk_u$ ← Generate a digital signature key pair
secret_shares [ ] ← Divide contact_info into $n$ secret pieces, e.g. using Shamir's secret
  sharing
$D_{auth}$ ← Pick the discovery node responsible for sending verification email
for $i \leftarrow 1, n$ do
    ▷ Algorithms used for these steps are set by the underlying anonymous communication
      network
    payload ← ($D_{auth}$, secret_shares[i], user_id, pk, $pk_u$)
    outgoing_path ← Randomly select a determined number of relay nodes
    outgoing_path.append(discovery_nodes[i])
    onion_header ← Create onion header from outgoing_path
    surb ← Generate SURB
    registration_msg ← Prepare a message using payload, onion_header, and surb, in the
      form that allows anonymous replies
    Pass registration_msg to the next node in the msg_path
end
Wait for a verification email until timeout limit $t_{verif}$
if Verification email received before $t_{verif}$ then
    Respond to the email
end
else
    Jump back to line 5
end
Wait for $n$ registration_msg responses until timeout limit $t_{reg}$
if $n$ registration_msg responses arrived before $t_{reg}$ then
    Success! End of registration
end
```

Algorithm 2: *Discovery node* message processing in ID-Verified Pudding

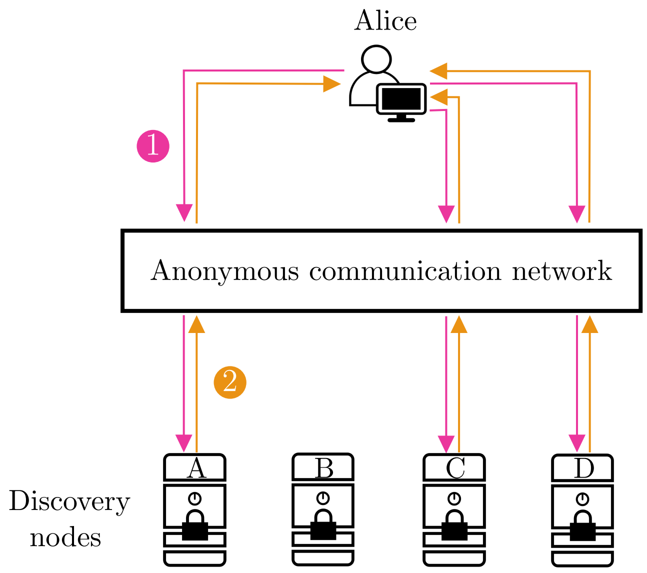
Input: A *msg* consisting of *payload*, *onion_header*, and *surb*

```
if msg is a registration message then
    $D_{auth}$, secret_piece, user_id, pk ← payload
    if A user record with user_id does not exist then
        random_challenge ← Generate random challenge
        if $D_{auth}$ is self then
            Wait for $n - 1$ random challenges from other discovery nodes
            Send a verification email to user_id including all the challenges
            Wait for a verification response email for timeout value $t_{verif\_resp}$
            if Verification response email received before $t_{verif\_resp}$ then
                if DKIM signature of the response email can be validated and contains
                  random_challenge then
                    Register the user user_registry[user_id] ← (secret_piece, pk)
                    Send the verification response email to all discovery nodes
                    Send the user confirmation message using surb
                end
                else
                    Drop the msg
                end
            end
        end
        else
            Send random_challenge to $D_{auth}$
            Wait for verification response email until timeout limit $t_{verif\_resp}$
            if Verification response email received before $t_{verif\_resp}$ then
                if DKIM signature of the response email can be validated and contains
                  random_challenge then
                    Register the user
                    Send the user confirmation message using surb
                end
                else
                    Drop the msg
                end
            end
        end
    end
    else
        Drop the msg
    end
end
```

4.3 User Discovery

To discover a user through ID-Verified Pudding, a searcher sends messages to all discovery nodes asking for the searchee's ID. In turn, each queried node uses the SURB attached to the message to respond with the contact information (γ) secret share it has for that ID (Figure 4.2). Since a (k, n) threshold secret sharing scheme is used for storing contact information, the searcher *only needs responses from k discovery nodes*, meeting the *availability* goal **(O4)**. Upon initial contact, the searchee can look up the searcher's ID on the system to verify her username and contact information, achieving *internal-identity-verification* goal **(S4)**. Using anonymous replies for discovery queries, these operations also meet the *unlinkability* goal **(S1)**.

User discovery in ID-Verified Pudding is described both in pseudocode (Algorithm 3) and through a step-by-step scenario below. The process is also outlined in Figure 4.2.



1 ● Alice randomly selects A, C, and D to query and sends Bob's ID to these

2 ● A, C, and D each send Bob's $(svk,\ secret_i)$ to Alice via SURB

Figure 4.2: Alice discovering Bob through ID-Verified Pudding. The system consists of discovery servers A, B, C and D, and employs a $(3, 4)$ threshold mechanism

When Alice searches for Bob through ID-Verified Pudding, where a (k, n) threshold scheme is deployed:

1. Alice creates n *discovery messages* in the form that allows anonymous replies **(S1)**. Payloads of all these messages are ID_{Bob}.
2. She sends these *discovery messages* out to the corresponding discovery nodes.
3. Upon receiving a *discovery message*, each discovery node D_i checks if it has a record for ID_{Bob} in its user registry.
   (a) If so, it uses the SURB attached to the *discovery message* to respond to Alice with Bob's username and the secret share mapped to it **(S1)**: $discovery\_msg_{D_i} \leftarrow (ID_{Bob}, secret_i)$.
   (b) Otherwise, the discovery node responds to Alice with an error code.
4. If all k responses arrive at Alice within a predetermined timeout period, she combines the secret shares to learn Bob's contact information and terminates the process.

Algorithm 3: User discovery in ID-Verified Pudding

```
$payload \leftarrow [\,]$
$picked\_discovery\_nodes \leftarrow$ Select $k$ random discovery nodes
for $node$ in $picked\_discovery\_nodes$ do
    $payload \leftarrow searchee\_id$
    $outgoing\_path \leftarrow$ Randomly select a determined number of relay nodes
    $outgoing\_path.append(node)$
    $onion\_header \leftarrow$ Create onion header from $outgoing\_path$
    $surb \leftarrow$ Generate SURB
    $discovery\_msg \leftarrow$ Prepare a message using $payload$, $onion\_header$, and $surb$
    Pass $discovery\_msg$ to the next node in the $msg\_path$
end
Wait for $k$ $discovery\_msg$ responses to return for timeout limit $t_{discovery\_timeout}$
if $k$ $discovery\_msg$ responses received before $t_{discovery\_timeout}$ then
    Combine the secret pieces to learn searchee's $contact\_info$
end
else
    Start over and retry
end
```

4.4 Updating User Data

ID-Verified Pudding employs a digital signature scheme to enable *authorised-user-data-updates* **(S3)**. To update some information on the discovery nodes, the user signs it with her secret account update key (sk_u) and sends the signature along with the data to every discovery node. The discovery nodes check the validity of the signature using the user's account update verification key (pk_u) and update the data as demanded if the signature is valid. This scheme can be described more systematically as below.

1. The user calculates n secret shares of her new contact information.
2. The user then signs each piece of secret with her secret account update key: $sig_i \leftarrow Sign(secret_i, sk_u)$.
3. The user creates and sends n *update request messages* which allow anonymous replies. Each of these messages include user ID, a payload, and the corresponding digital signature: $update\_request\_secret_i \leftarrow (ID, secret_i, sig_i)$.
4. Upon receiving an *update request message*, each discovery node D_i verifies the signature using the user's pk_u: $(SigVerify(pk_u, secret_i, sig_i))$.
   (a) If the signature is valid, the discovery node updates the user's information as requested and uses the SURB to send a confirmation message to the user.
   (b) Otherwise, it drops the *update request message*.

This mechanism meets the *availability* goal, because as long as k or more discovery nodes have the updated information, the user remains discoverable. If some discovery nodes are offline during the update process under this assumption, the user device periodically retries updating information at these nodes.

It is also worth noting that since the system maintains *unlinkability* (S1), it cannot keep track of who has outdated information about the users and cannot notify them directly. Instead, users who perform an update can message the users on their address book via the underlying anonymous communication network to inform them.

4.5 Account Recovery

Thanks to the *external-identity-verification* feature, if an ID-Verified Pudding user loses her secret account update key, she can regain access to her account via email (sk_u). This process is considerably similar to the registration process (Section 4.2). The only difference is that that the **Update Preparation** stage described below replaces the **Preparation (Section 4.2.A)** stage:

A. Update Preparation

When a user has lost their sk_u, and therefore wants to recover her Pudding account:

1. The user arbitrarily picks a discovery node as D_{auth}. This node bears the responsibility of exchanging emails with the user during the account recovery process.

2. She also generates a *new* account update key pair (pk_{u_new}, sk_{u_new}).

3. The user sends an *account recovery message* to each discovery node D_i in the form that allows anonymous replies. Each message includes the user's ID, an indicator showing that the user wants to recover the Pudding account tied to this ID, as well as D_{auth}'s identifier:
   $account\_recovery\_msg_i \leftarrow (D_{auth}, ID, pk_{u\_new}, account\_recovery\_flag)$

For the remainder of the account recovery process, this stage is followed by the stages **Verification (Section 4.2.B)** and **Registration and Confirmation (Section 4.2.C)**. Indeed, the term *registration* should be changed to *update.*

It is worth mentioning that this account recovery mechanism meets the *availability* goal, since it allows the user to regain access to her account as long as k or discovery nodes are available.

4.6 Tradeoffs and Limitations

This section describes the four essential shortcomings of ID-Verified Pudding: It fails to meet the *username-anonymity* **(S3)** and *membership-unobservability* **(S6)** goals; uses email as the only *external-identity-verification* **(S5)** channel, which might cause vulnerabilities; and it meets *contact-information-privacy* **(S2)** under specific conditions.

No *username-anonymity* (S3): As mentioned earlier in Section 3.4, ID-Verified Pudding does not provide *username-anonymity*, because this goal is in conflict with *external-identity-verification* goal (S5). This single notable privacy deficiency of ID-Verified Pudding must not be overlooked, since *username-anonymity* is an essential goal for the users who are concerned about keeping their usernames hidden from the system. The second type of Pudding, Incognito Pudding sets to serve as a remedy for ID-Verified Pudding's this main deficiency (Chapter 5).

No *membership-unobservability* (S6): An adversary who wants to tell if a username belongs to a member of the network or not can assume two different roles in the Pudding network: If the adversary (1) controls a discovery node, it can easily see the whole list of registered usernames; (2) as a user, it might perform *crawling*, as described in Section 3.2. Although rate limiting might be employed to mitigate this issue, as demonstrated by a recent study, [13] it is problematic to employ rate limiting to prevent crawling while maintaining functionality. Another mitigation technique to crawling might be responding with realistic dummy user data when queried with an unregistered username. Although the first adversarial capability is inherent to this protocol and therefore unavoidable, the mentioned mitigation strategy against malicious users might still be useful in certain settings.

Email as the Only Form of *external-identity-verification*: Li et al. showed that using emails for verification and account recovery among different services creates a single point of failure. The same study also demonstrated that many leading email service providers do not make enough effort to protect email addresses [76]. However, email address compromise is a much larger issue than its consequences in ID-Verified Pudding.

Specific Conditions for *contact-information-privacy* (S2): As indicated previously in Section 3.3, it is important to once again acknowledge that *contact-information-privacy* is subject to the assumption that threshold-many or more discovery nodes do not collude. Moreover, even under non-collusion assumptions, a rogue discovery node can make legitimate queries using the IDs in its user registry to learn contact information associated with them.

4.7 Summary

This chapter has described ID-Verified Pudding, one of the two Pudding implementations presented in this dissertation. This Pudding variant allows users to register to the system with an email address and verify that they have access to that email address using DKIM. Through this feature, users can discover others using their email addresses and be confident that they are who they claim to be. In other words, ID-Verified Pudding transfers the practicality and trust that email provides to anonymous communication networks.

Chapter 5

Incognito Pudding

This chapter presents Incognito Pudding, the second Pudding implementation presented in this dissertation. Storing specially salted hashes of usernames in the discovery nodes, this implementation allows users to use Pudding *incognito*, i.e. keep their usernames hidden from the discovery nodes. This chapter begins by presenting an overview of the Incognito Pudding protocol, listing the system goals it aims to meet, and explaining its core idea, which is OPRF-salted hashing. Following this section, it presents a detailed illustrative scenario, and moves on to describing the protocol's functionalities, namely registration, user discovery, and user data updates. Finally, the chapter finishes by describing the limitations of the protocol.

5.1 Protocol Overview

Incognito Pudding keeps usernames secret from the discovery nodes, achieving *username-anonymity* **(S3)**. This has great importance in private user discovery, because several studies have shown that usernames, even without any additional information, can be used for linking user presence across different online platforms, which can lead to tying those to real-world identities as well [77, 78, 79, 80]. Being a serious privacy threat in the larger context, user identification is particularly undesirable for users of an anonymous communication network, since they presumably seek superior anonymity compared to users of non-anonymous communication systems. Even though *crawling* remains as an issue, preventing usernames from being easily accessible to discovery nodes in large quantities is still valuable. To hide usernames from the system, Incognito Pudding does not store usernames but their salted hashes created using the particular method explained in Section 5.1.2.

Keeping usernames hidden from the discovery nodes comes at the cost of foregoing the ability to tie user presence within Pudding to email addresses as external identifiers. This is because the user is not able to prove to the discovery nodes that an identifier that allows communication through external channels, such as an email address, belongs to her without revealing information about the identifier.

5.1.1 System Goals Analysis

Incognito Pudding achieves all *Operational Goals.* Regarding *Security and Privacy*, it reaches all goals except from two, namely *external-identity-verification* **(S5)** and *membership-unobservability* **(S6)**, because in this context, these goals are in conflict with Incognito Pudding's fundamental goal, *username-anonymity* **(S3)**. The protocol achieves the remaining *Security and Privacy Goals*, namely *unlinkability* **(S1)**, *contact-information-privacy* **(S2)**, and *internal-identity-verification* **(S4)** the same way as ID-Verified Pudding (see Section 4.1.1).

In terms of *Usability*, this protocol achieves all goals but *account-recovery* **(U4)**, since this goal cannot be optimally realised simultaneously with *username-anonymity* (a discussion about suboptimal ways to achieve this can be found in Section 5.6). Incognito Pudding allows the users to remain within the system for all operations, achieving *all-internal-execution* **(U1)**; supports picking any string as username, enabling *human-memorable-username-selection* **(U2)**; and allows *authorised-user-data-updates* **(U3)** using digital signatures.

5.1.2 The Core Idea: Salt Creation and Username Hashing

This section explains the core idea behind Incognito Pudding, which is a special salt creation and salted hashing method.

Hashing is not enough: Incognito Pudding's main goal is hiding usernames of registered users from the discovery nodes. One common approach to storing information in an obfuscated form is storing the hash of it. However, storing simple hashes of usernames is not sufficient to meet the desired level of *username-anonymity.* A key study by Perito et al. showed that although the necessity for usernames to be unique within a platform may drive users to pick high-entropy usernames, users tend to choose similar usernames across different platforms [77]. Therefore, storing hashes of usernames in Incognito Pudding would be prone to offline attacks because not only usernames might consist of words found in a dictionary, but also they are mostly related to other usernames of users, which might be publicly available. This would potentially cause a major privacy risk since Pudding identities could even be linked to real-world identities through other identities in different platforms.

Regular salted hashing is not suitable: The described problem with storing hashed usernames might seem similar to that with passwords. Therefore, using the same commonly used solution, i.e. creating distinct arbitrary *salt* values, might come to mind as a cure. However, this solution cannot be applied to these two problems in the same way. In password hashing, salted hash values are linked to plaintext usernames. This allows the discovery nodes to search their databases for the asked username and find the password hash mapped to it to do the necessary checks. On the other hand, in Incognito Pudding, since the usernames should be kept private, salting them would make it impractical to find them in a database without any searchable identifier.

Using OPRF to generate salts: To create searchable identifiers from usernames without compromising their privacy, Incognito Pudding uses OPRF with an approach that is similar to password hardening case presented in Section 2.3.4. The main idea is deriving strong secrets from low-entropy data in an unlinkable, irreversible, yet reproducible way. To achieve this, at least threshold-many discovery nodes and a searcher evaluate OPRFs. In each evaluation, one discovery node is the *key holder* with a distinct secret key (k), and the searcher is the *input provider* supplying searchee's username as input (x). Once the searcher obtains separate values (y) from OPRF evaluations with the discovery nodes, she uses these values as salts to compute a hash value of the username and query the discovery nodes with this value.

Note that in this setting, all searchers compute the same salts with the same discovery nodes for a given username. Inversely, a searcher who does not know the username cannot generate the salts, and therefore cannot compute the salted hashes. Moreover, knowing the username is not enough to compute the salt values unless the OPRF keys secretly stored in discovery nodes are exposed. In addition, the pseudorandomness of these salt values means that they are irreversible, i.e. one cannot feasibly figure out a username from its salts. Since each username is salted with multiple salt values from different discovery nodes' OPRF evaluations, a discovery cannot brute-force search usernames running the OPRF protocol with its own key either. Ultimately, this approach allows usernames to be hidden from the discovery nodes, but still discoverable by those who know them.

Generating *handles*: Incognito Pudding uses the term handle to refer to salted hashes of usernames.

Definition 5.1.1. Handle: A handle is a salted hash of a username, which is used as an input to Incognito Pudding's user discovery function.

In other words, handles are mapped to the contact information to create key-value pairs. As Incognito Pudding uses (k, n) threshold secret sharing scheme, having a single handle corresponding to a username can be achieved through either (a) all discovery nodes having the same key to use in salt generation through OPRF, or (b) users querying all n discovery nodes for user discovery. Both these options are problematic: The first option diminishes the number of distinct salts per username from n to 1, which increases the chance of malicious discovery nodes brute-force retrieving plaintext usernames from hashes. The second option, on the other hand, overthrows the practical advantage of querying $k < n$ nodes being sufficient.

For this reason, each username is represented with multiple (specifically, $\binom{n}{k}$-many) handles instead of a single one. Each of these handles is a k-many-times salted hash of the username so that *there exists one handle for each of the k-combinations of the n discovery nodes.* Each discovery node D_i receives the handles for the combinations that include D_i. This allows the users to pick any of the k discovery nodes for user discovery.

Moreover, to prevent colluding servers from being able to match handles among each other, users hash each handle with a known unique value for each discovery node, such as a node name, before submitting it to the discovery nodes. The resulting *unique handles* allow the user to be represented with distinct identifiers among all discovery nodes.

5.2 An Illustrative Example

This section puts the OPRF salt computation and handle generation processes in context by presenting an example scenario with registration and user discovery in Incognito Pudding. It is important to note that the registration and discovery mechanisms are described in detail in Sections 5.3 and 5.4. This section only previews these mechanisms with the intention of demonstrating the core idea behind Incognito Pudding, as specified in Section 5.1.2. The scenario below happens in a system that employs a $(3, 4)$ threshold secret sharing scheme and has discovery nodes (A, B, C, D).

Registration of a user:

1. The user creates a public-private key pair and determines her ID as $abc123@cam.ac.uk$.

2. Using OPRF, the user contacts each of the discovery nodes to generate 4 salts; $salt_A, salt_B, salt_C, salt_D$:

$$\begin{aligned} salt_A &\longleftarrow OPRF(A, \text{`}\texttt{abc123@cam.ac.uk}\text{'}) \\ salt_B &\longleftarrow OPRF(B, \text{`}\texttt{abc123@cam.ac.uk}\text{'}) \\ salt_C &\longleftarrow OPRF(C, \text{`}\texttt{abc123@cam.ac.uk}\text{'}) \\ salt_D &\longleftarrow OPRF(D, \text{`}\texttt{abc123@cam.ac.uk}\text{'}) \end{aligned}$$

3. The user generates $\binom{n}{k} = \binom{4}{3} = 4$ handles using different combinations of the generated salts.

$$\begin{aligned}
handle_1 &\longleftarrow H(salt_A \parallel salt_B \parallel salt_C \parallel \text{`}\texttt{abc123@cam.ac.uk}\text{'}) \\
handle_2 &\longleftarrow H(salt_A \parallel salt_B \parallel salt_D \parallel \text{`}\texttt{abc123@cam.ac.uk}\text{'}) \\
handle_3 &\longleftarrow H(salt_A \parallel salt_C \parallel salt_D \parallel \text{`}\texttt{abc123@cam.ac.uk}\text{'}) \\
handle_4 &\longleftarrow H(salt_B \parallel salt_C \parallel salt_D \parallel \text{`}\texttt{abc123@cam.ac.uk}\text{'})
\end{aligned}$$

4. The user then computes $\binom{n-1}{k-1} = \binom{3}{2} = 3$ hashes for *each* of these values by hashing them with the name of each discovery node. As a result, the user has $[\binom{n-1}{k-1} \cdot \binom{n}{k}]$-many *unique handles*:

$$\begin{aligned}
u\_handle_{1_A} &\longleftarrow H(A \parallel handle_1) \\
u\_handle_{1_B} &\longleftarrow H(B \parallel handle_1) \\
u\_handle_{1_C} &\longleftarrow H(C \parallel handle_1) \\
u\_handle_{2_A} &\longleftarrow H(A \parallel handle_2) \\
u\_handle_{2_B} &\longleftarrow H(B \parallel handle_2) \\
u\_handle_{2_D} &\longleftarrow H(D \parallel handle_2) \\
u\_handle_{3_A} &\longleftarrow H(A \parallel handle_3) \\
u\_handle_{3_C} &\longleftarrow H(C \parallel handle_3) \\
u\_handle_{3_D} &\longleftarrow H(D \parallel handle_3) \\
u\_handle_{4_B} &\longleftarrow H(B \parallel handle_4) \\
u\_handle_{4_C} &\longleftarrow H(C \parallel handle_4) \\
u\_handle_{4_D} &\longleftarrow H(D \parallel handle_4)
\end{aligned}$$

5. The user secret shares her contact information along with $\binom{n-1}{k-1} = \binom{3}{2}$ unique handles to each discovery node. The set of unique handles each discovery node receives consists of all the values generated using the handles that include the salt that this discovery node has generated. The discovery nodes consequently add the user to their registers as follows:

$$\begin{aligned}
&A \textit{ registers } (u\_handle_{1_A}, u\_handle_{2_A}, u\_handle_{3_A}) : secret\_share_A \\
&B \textit{ registers } (u\_handle_{1_B}, u\_handle_{2_B}, u\_handle_{4_B}) : secret\_share_B \\
&C \textit{ registers } (u\_handle_{1_C}, u\_handle_{3_C}, u\_handle_{4_C}) : secret\_share_C \\
&D \textit{ registers } (u\_handle_{2_D}, u\_handle_{3_D}, u\_handle_{4_D}) : secret\_share_D
\end{aligned}$$

Once the user's registration is complete, a searcher discovers the user with username *abc*123@*cam.ac.uk* as follows:

1. The searcher arbitrarily selects A, B, and C as the $k = 3$ discovery nodes to query.

2. She then generates the 3 salt values for *abc*123@*cam.ac.uk* through performing OPRF with every discovery node $(salt_A, salt_B, salt_C)$:

$$\begin{aligned} salt_A &\longleftarrow F_{k_A}(\text{`}\texttt{abc123@cam.ac.uk}\text{'}) \\ salt_B &\longleftarrow F_{k_B}(\text{`}\texttt{abc123@cam.ac.uk}\text{'}) \\ salt_C &\longleftarrow F_{k_C}(\text{`}\texttt{abc123@cam.ac.uk}\text{'}) \end{aligned}$$

3. Using these salts, the searcher then calculates the handle for this combination of discovery nodes:

$$handle \longleftarrow H(salt_A \parallel salt_B \parallel salt_C \parallel \text{`}\texttt{abc123@cam.ac.uk}\text{'})$$

4. Then, the user calculates 3 unique handles by hashing the handle with the name of each selected discovery node:

$$\begin{aligned} u\_handle_A &\longleftarrow H(A \parallel handle) \\ u\_handle_B &\longleftarrow H(B \parallel handle) \\ u\_handle_C &\longleftarrow H(C \parallel handle) \end{aligned}$$

5. The searcher queries each discovery node with the corresponding unique handle.

6. Each discovery node A, B and C correspondingly searches in its register u_handle_A, u_handle_B and u_handle_C. As they are the same as $u\_handle_{1_A}$, $u\_handle_{1_B}$ and $u\_handle_{1_C}$ of Step 4, respectively, they successfully find the secret pieces $secret\_share_A$, $secret\_share_B$, $secret\_share_C$ and send them back to the user using the SURB received along with the *discovery message.*

7. The user combines the secrets to learn contact information of the searchee.

5.3 Registration to the System

Incognito Pudding's registration phase provides the preconditions for achieving some of the system goals. To meet the protocol's essential goal, *username-anonymity* **(S3)**, the registering user follows the OPRF salt calculation, as well as the handle generation and distribution logic described in Section 5.1.2. Similar to ID-Verified Pudding, the user secret-shares her contact information with the discovery users to achieve *contact-information-privacy* **(S2)**, and includes digital signature key generation and sharing for *authorised-user-data-updates* **(S3)**.

Below is a step-by-step description of the registration process for a user. How registration takes place in practice is also illustrated with an example in Section 5.2 and with pseudocode in Algorithms 4 and 5.

The registering user

1. determines an ID, which can be any string.
2. creates a public-private key pair (pk, sk) and an account update key pair $(pk_u,\ sk_u)$ **(S3)**.
3. splits her contact information (γ) into n secret shares **(S2)**.
4. obtains *at least k salts* from the ID by performing an OPRF with each discovery node. If some discovery nodes are unavailable during this process, the user can still register as long as k or more discovery nodes are available **(O4)**. In this case, the user device keeps track of the unavailable discovery nodes and continues the registration with them when they are serviceable again.
$$salt_{D_i} \longleftarrow F_{k_{D_i}}(ID)$$
5. computes $\binom{n}{k}$ handles **(S3)**.
$$handle_i \longleftarrow H(salt_a \parallel salt_b \parallel salt_c \parallel \ldots \parallel ID)$$
for each k-combination $(D_a, D_b, D_c, \ldots)$ of n discovery nodes.

6. prepares n *registration messages* in the form that allows anonymous replies. The *registration messages* have the following properties:

   - Each message carries a set of $\binom{n-1}{k-1}$-many unique handles, a secret piece, and the user's pk.
   - Each set of unique handles per discovery node D_i is computed by hashing each handle that has the salt generated with D_i with a known unique value tied to D_i, e.g. its node name.

     $$u\_handle\_set_i \leftarrow H(D_i \parallel handle \mid handle\ contains\ D_i\text{'s salt})$$

   - Each message payload is constructed as:

     $$payload_i \leftarrow (secret_i, u\_handle\_set_i, pk, pk_u)$$

7. Sends out the prepared *registration messages* to the discovery nodes.

Upon receiving a *registration message*, each discovery node D_i

1. checks if a user record already exists with the received $u\_handle\_set_i$.
   (a) If this is the case, it drops the message.
   (b) Otherwise, it adds the new user to its user registry, mapping the unique handles to the secret piece and public keys:
   $u\_handle\_set_i : (secret_i, pk, pk_u)$
2. uses the SURB attached to the *registration message* to create a new message directed to the user without knowing her identity. This message lets user know that her registration at D_i was successful.

The registration process completes when the user receives confirmation messages from at least k discovery nodes within a predetermined timeout period. As with ID-Verified Pudding, better usability can be achieved by tracking these messages automatically at the user device.

Algorithm 4: Registering to Incognito Pudding as a user

```
1  $user\_id \leftarrow$ Pick an owned email address or any string
2  $pk, sk \leftarrow$ Generate a public-private key pair for encryption-decryption
3  $pk_u, sk_u \leftarrow$ Generate an account update key pair
4  $secret\_shares$ [ ] $\leftarrow$ Divide $contact\_info$ into $n$ secret pieces
5  $salts \leftarrow$ Empty dictionary
6  $discovery\_node\_combinations$ [ ] $\leftarrow$ $k$-combinations of the $discovery\_nodes$
7  for $discovery\_node$ in $discovery\_nodes$ do
8      $salts[discovery\_node] \leftarrow OPRF(self.ID, discovery\_node)$
9  end
10 for $discovery\_node$ in $discovery\_node$ do
11     $combinations\_with\_discovery\_node$ [ ] $\leftarrow$ All combinations in
         $discovery\_node\_combinations$ containing $discovery\_node$
12     $u\_handles$ [ ] $\leftarrow$ Empty array
13     for $combination$ in $combinations\_with\_discovery\_node$ do
14         $salt\_combination \leftarrow$ Values in $salts$ that have elements of $combination$ array as
             keys, concatenated in deterministic order (e.g. lexicographic order by discovery
             node name) to form a string
15         $u\_handle \leftarrow Hash(discovery_n ode.name \parallel Hash(salt\_combination \parallel self.ID))$
16         $u\_handles.append(u\_handle)$
17     end
18     $payload \leftarrow (secret\_shares.pop(), u\_handles, pk, pk_u)$
19     $outgoing\_path \leftarrow$ Randomly select a determined number of relay nodes
20     $outgoing\_path.append(discovery\_nodes[i])$
21     $onion\_header \leftarrow$ Create onion header from $outgoing\_path$
22     $surb \leftarrow$ Generate SURB
23     $registration\_msg \leftarrow$ Prepare message using $payload$, $onion\_header$, and $surb$, in the
         form that allows anonymous replies
24     Pass $registration\_msg$ to the next node in the $msg\_path$
25 end
26 Wait for $n$ $registration\_msg$ responses until timeout limit $t_{reg}$
27 if $n$ $registration\_msg$ responses arrived before $t_{reg}$ then
28     Success! End of registration
29 end
```

Algorithm 5: *Discovery node* message processing in Incognito Pudding

```
Input: A msg consisting of payload, onion_header, and surb
if msg is a registration message then
    secret_piece, u_handles, pk ← payload
    if No user record with any of the handles in u_handles i found then
        Register the user, i.e. user_registry[u_handles] ← (secret_piece, pk)
    end
    else
        Drop the msg
    end
end
else if msg is a discovery message then
    searchee_id ← payload
    if A user record with user_id found then
        Using surb, respond to the user with (searchee_id,secret_piece)
    end
    else
        Drop the msg
    end
end
```

5.4 User Discovery

In Incognito Pudding, to discover a searchee, a searcher computes salt values for the searchee's username with any k discovery nodes using an OPRF, and computes a single salted hash value using these salts. The searcher can then hash this value once again for each selected discovery node with the discovery node's fixed known value, such as its name. The resulting unique handles can be used by the searcher to discover the searchee.

As in ID-Verified Pudding, the user discovery phase achieves *unlinkability* goal **(S1)** and *internal-identity-verification* **(S4)**. This phase also reaches the *availability* **(O4)** goal, since users can discover others as long as k or more discovery nodes are functional.

Steps of the user discovery protocol in a scenario where Alice is the searcher and Bob is the searchee are described below. Additionally, Algorithms 5 (line 10 - 18) and 6 explain the process in pseudocode.

When Alice searches for Bob through Incognito Pudding, where a (k, n) threshold scheme is deployed.

1. Alice randomly picks k discovery nodes.
2. She then obtains k salts for Bob's ID performing an OPRF with each discovery node. This process happens over the anonymous communication network, using anonymous replies.
3. She computes Bob's *handle* for the selected set of k discovery nodes, hashing his ID with each of them:

   $$handle \longleftarrow H(salt_a \parallel salt_b \parallel salt_c \parallel \ldots \parallel ID_{Bob})$$

   for the selected k-combination of n discovery nodes: $(D_a, D_b, D_c, \ldots)$
4. Alice creates k *discovery messages* in the form that allows anonymous replies. Each discovery node's message payload is a unique handle (*u_handle*), prepared by hashing *handle* with the discovery node's name or another known unique value.
5. Alice then sends *discovery messages* to the k selected discovery nodes.
6. Upon receiving the discovery message, each discovery node D_i searches for a user entry with *u_handle* in its user registry.
   (a) If it has a matching user record, it uses the SURB attached to the *discovery message* to respond to Alice with the secret share it has along with the *u_handle*: $payload_{D_i} \leftarrow (u\_handle, secret_i)$
   (b) Otherwise, it drops the message and responds to Alice with a suitable error code.
7. If all k responses arrive at Alice within a predetermined timeout period, she combines the secret shares to reveal Bob's contact information. Otherwise, she retries the whole process picking different nodes than the non-responding ones.

This mechanism meets the *availability* goal, since Bob is discoverable as long as k discovery nodes are available. Moreover, it also supports *internal-identity-verification*, since Bob can repeat the discovery process for Alice to confirm her internal identity upon initial contact.

Algorithm 6: User discovery in Incognito Pudding

```
payload ← [ ]
picked_discovery_nodes ← Select $k$ random discovery nodes
salts ← Empty dictionary
for discovery_node in picked_discovery_nodes do
    salts[discovery_node] ← $F_{k_{discovery\_node}}(searchee\_id)$
end
handle = Hash(salt_combination ∥ searchee_ID))
for discovery_node in picked_discovery_nodes do
    payload ← Hash(discovery_node.name ∥ handle))
    outgoing_path ← Randomly select a determined number of relay nodes
    outgoing_path.append(discovery_node)
    onion_header ← Create onion header from outgoing_path
    surb ← Generate SURB
    discovery_msg ← Prepare an anonymous reply message using payload, onion_header,
      and surb
    Pass discovery_msg to the next node in the msg_path
end
Wait for $k$ discovery_msg responses to return for timeout limit $t_{discovery\_timeout}$
if $k$ discovery_msg responses received before $t_{discovery\_timeout}$ then
    Combine the secret pieces to learn searchee's contact_info
end
else
    Start over and retry
end
```

5.5 Updating User Data

Incognito Pudding's mechanism for *authorised-user-data-updates* **(U3)** is essentially the same as ID-Verified Pudding's (Section 4.4): It uses digital signatures to authorise updates to user data. As a reminder, the update mechanism works as follows: When a user wants to amend any data, she digitally signs the new data with her sk_u, and submits the signature along with the data. The discovery nodes use the user's pk_u to validate the signature and update her data as requested, if the signature is valid.

5.6 Tradeoffs and Limitations

The apparent shortcomings of Incognito Pudding are as follows: This protocol does not support *external-identity-verification* **(S5)**, *membership-unobservability* **(S6)**, and *account-recovery* **(U4)** goals. Moreover, the *crawling* issue discussed in Sections 3.2 and 4.6 affects Incognito Pudding as well. This section discusses these limitations in detail.

No *external-identity-verification* (S5): Incognito Pudding's main goal, *user name-anonymity*, prevents the discovery nodes from linking user accounts to email addresses as a form of verified external identity. This is because one simply has to know an email address to be able to send emails to it. Likewise, to learn the public verification key for verifying a DKIM signature, a discovery node has to make a DNS query using the sender's email address or some identifying information about it, such as its domain.

Having a *trusted verification server* to handle the email verification can allow *external-identity-verification*. However, this solution creates a single point of failure in the system. Still, this may be a compromise that Incognito Pudding system providers and users might be willing to make in certain settings.

Crawling: Although Incognito Pudding hides usernames from discovery nodes, it still does not provide *membership-unobservability* **(S6)**, because *crawling* remains a problem. As mentioned in Section 4.6, *contact-information-privacy* **(S2)** is also subject to the *crawling* issue. In spite of these shortcomings, preventing malicious discovery nodes from learning registered usernames in bulks is a valuable privacy factor, as noted earlier in Section 5.1.

No *account-recovery* (U4): Since Incognito Pudding does not support *external-identity-verification* **(S5)**, the discovery nodes do not have an external channel to access a user and verify her identity to grant access to her account. Although allowing account recovery through so-called security questions is possible, both research [81, 82] and anecdotal evidence (a well-known example being the Sarah Palin email hack [83]) indicate that these questions are vulnerable to account hijacking. A second alternative for allowing *account-recovery* might be following a method similar to Facebook *trusted contacts* [84], who can collectively help recover a Facebook user's account. However, the risk of maliciously gaining control of a user account through phishing and social engineering attacks must be noted.

5.7 Summary

This chapter has described the second Pudding protocol that this dissertation presents, namely Incognito Pudding. This protocol design allows the user to register to the system without revealing her username. However, this core aim is in conflict with *external-identity-verification*, *membership-unobservability*, and *account-recovery* goals. Despite these tradeoffs, hiding usernames from the discovery nodes is a valuable goal, as studies show that usernames can be used for linking user identities accross platforms [77, 78, 79, 80], which might be a particularly critical privacy concern for the users of an anonymous communication network.

Chapter 6

Implementation

Thus far, this dissertation has presented ID-Verified and Incognito Pudding protocols theoretically. To empirically evaluate these protocols and substantiate their practicality and availability promises, I have implemented a simulation tool. The simulations tested by this tool cover all possible runs of the protocols in a finite state space. This chapter describes the capabilities and limitations of this implementation.

6.1 Pudding Simulation Tool

ID-Verified Pudding (Chapter 4) and Incognito Pudding (Chapter 5) protocols claim to achieve practicality by meeting non-conflicting subsets of the ideal discovery system's goals (Section 3.3). Both protocols promise *liveness*, since all operations remain functional as long as at least threshold-many discovery nodes are available. The protocols also aim *completeness*, fully describing the processes they involve. The main objective of this simulator is to demonstrate the practicality of the two Pudding protocols by testing their liveness and completeness with a *model checking* approach.

Model checking is a formal analysis method, where all possible inputs of a system are automatically traversed and all possible outputs are checked for a finite state space [85]. This simulation tool functions as a model checker, since it aims to check all possible runs of the two Pudding protocols in a state space bounded by the parameters presented in Table 6.1 and explained in the next section. It is worth mentioning that this simulator does not intend to formally verify the protocols nor measure their performance.

In terms of technicalities, this tool is a discrete-event simulator that runs on a single machine and represents all actors and messages as objects. This object-oriented approach enables effective examination of the relationships between entities. The Pudding simulator is written in Python language and uses PyCryptoDome package to implement the main cryptographic primitives. The simulator represents time with an event-driven tick-based approach, where every request-response round between a user and a discovery node is executed within a tick.

As the main intention of the simulation tool is to demonstrate practicality, it uses abstractions to hide the details of the actions whose inner workings do not affect the logic of the protocols. For instance, the actions of email sending and DKIM signature checking are treated as black boxes. Similarly, OPRF is simplified as a function that takes a username as an input and returns a pseudorandom string using the username as a seed.

6.2 Evaluation

The developed simulation tool tests the core protocol functionalities, i.e. registration, discovery, and user data update, in a comprehensive set of scenarios. To cover scenarios with all possible state configurations in a determined state space, I implemented a scenario generator. This state space is bounded by the values of five parameters listed Table 6.1, namely Pudding type, number of users, number of discovery nodes, threshold value, and user action. The generator follows separate approaches for generating scenarios for evaluating completeness and liveness of the protocols.

Parameter	**Possible values for *completeness***	**Possible values for *liveness***
Num. of users (`U`)	1 <`U` <3	1 <`U` <2
Num. of discovery nodes (`N`)	1 <`N` <5	1 <`N` <4
Threshold value (`K`)	1 <`K` <`N`	1 <`K` <`N`
User action	`REGISTRATION`, `UPDATE`, `DISCOVERY`	
Pudding type	`ID_VERIFIED`, `INCOGNITO`	

Table 6.1: The parameters used for testing completeness and liveness of the protocols and their possible values.

6.2.1 Completeness Evaluation

Completeness evaluation aims to demonstrate that the protocols represent the system behaviour for all possible user behaviours. To achieve this, it investigates the scenarios where the users may behave arbitrarily, assuming that all discovery nodes are available.

The set of scenarios used for evaluating completeness consists of all possible combinations and orderings of user actions under each parameter configuration presented in Table 6.1. In a logical sense, these scenarios can be regarded as collections of several success and failure cases, which can be grouped under the categories listed in Table 6.2. For instance, a successful scenario can be as follows: *Alice registers to the system, Bob registers to the system, Bob updates his user data*; or SI, SI, SII. Changing the ordering of the second and third events results in a different scenario represented with cases SI, FII, SI, and is expected to fail. Despite their simplicity, these cases and parameters are comprehensive and expressive to enough to generate 37,080 completeness test scenarios.

6.2.2 Liveness Evaluation

Liveness evaluation aims to demonstrate that the discovery mechanism maintains its functionality as long as at least threshold-many discovery nodes are available within a timeout period. The scenarios for this evaluation assume that discovery nodes may deviate from the protocol, while users attempt to legitimately register, update user data, and discover a user following the protocols. To achieve this,

Code	Success case
SI	A user registers to the system
SII	A registered user attempts to update
SIII	A registered searcher attempts to discover a registered searchee

Code	Failure case
FI	Multiple users attempt to register with the same username
FII	An unregistered user attempts to update her data
FIII	A registered searcher attempts to discover an unregistered searchee
FIV	An unregistered searcher attempts to discover an registered searchee
FV	An unregistered searcher attempts to discover an unregistered searchee

Table 6.2: Categories of success and failure cases in auto-generated completeness test scenarios.

liveness scenarios span all possible configurations of discovery nodes independently becoming available or unavailable at any point of execution for every parameter configuration, within a timeout period determined by the value of `N`.

The described configurations result in 7492 liveness scenarios. The reason why this number is smaller when compared to completeness tests is that increasing the `K` and `N` value ranges escalate the number of scenarios exponentially. For instance, incrementing the current upper boundary of `N` by one results in more than one million (1,195,876) scenarios. Therefore, liveness parameters were adjusted to be able to test protocol runs exhaustively within a reasonable time period.

6.3 Limitations

As mentioned earlier in this chapter, this simulation tool should not be treated as a formal verification tool. In other words, although this simulator is helpful in finding errors in a Pudding protocol, the lack thereof does not guarantee correctness. Besides, this tool does not aim to evaluate security or performance of the protocols.

It is also worth noting that the scenarios tested in this implementation is limited both in ranges and types of parameters due to time constraints. For instance, possible failure cases in email verification could not be investigated. Chapter 7 provides some suggestions for further research in these areas. Despite its exploratory nature, this simulator offers concreteness for the claims of the Pudding protocols.

6.4 Summary and Results

The simulation tool presented in this chapter aims to demonstrate that the two Pudding protocols are practical by testing their liveness and completeness promises. As a result of testing all possible protocol runs in a total of 44,572 scenarios, *the protocols behaved as expected in all of them.* Putting the full logic of the protocol into action through this tool, it is apparent that in a large number of scenarios, both Pudding protocols are realistic and functional.

Another significant outcome of this simulation tool is that it has resulted in an improvement in one of the protocols. Initially, to discover a user through ID-Verified Pudding, the searcher would send discovery requests to k discovery nodes and retry discovery with a different combination if some of them are unavailable. Liveness tests demonstrated that it is simpler and more time-efficient to send discovery requests to all discovery nodes instead and terminate the process once threshold-many responses are achieved. Therefore, the protocol was revised according to this finding.

Chapter 7

Summary and Conclusions

Anonymous communication networks can allow messaging with metadata privacy, providing better privacy than popular encrypted messaging applications. However, these networks currently lack a usable and privacy-preserving mechanism that enables users to discover others on the network, which is a barrier to the adoption of anonymous communication networks for messaging, as previous research suggests [1]. This dissertation is the first study to explore the whole landscape of this acute yet underexplored problem.

There are several important areas where this study makes an original contribution to the literature. One is proposing an ideal mechanism for user discovery that is both usable and privacy-preserving. Regarding usability, this mechanism allows users to search for others through human-memorable usernames without leaving the anonymous communication network. In terms of privacy, it essentially aims to (1) allow retrieval of the user information needed to contact them, (2) while keeping usernames private from the discovery mechanism, and (3) making the whole process unlinkable in the sense that the discovery mechanism cannot figure out *who is searching for whom.*

Another contribution of this dissertation is analysing the ideal mechanism's system goals and describing its threat and trust assumptions. This analysis has identified that some properties of the ideal user discovery mechanism are in conflict in practice. Therefore, this study has established two concrete user discovery protocols which achieve different subsets of the the ideal mechanism's goals and represent different points in the usability-privacy tradeoff space.

The first protocol, ID-Verified Pudding, allows users to use email addresses as usernames. Using DKIM, this protocol links the user's identity within the discovery mechanism to an external identity, email address. This provides the searchers confidence that the user they discover is who she claims to be. Besides these advantages, ID-Verified Pudding makes usernames known to discovery nodes, which might be undesirable for users who require to hide that they are a member of the anonymous messaging system.

The second protocol, Incognito Pudding, aims to provide better anonymity by hiding usernames from discovery nodes, while allowing discovery without the need to know anything but the username. However, to achieve this level of anonymity, Incognito Pudding sacrifices the ability to link the user's internal identity to an external one.

Through the second protocol design, this dissertation has applied Oblivious Pseudorandom Functions (OPRFs) to the context of discovery for the first time. One existing use area of OPRFs is hardening human-memorable passwords by deriving strong secrets from them. I have adapted this idea to privacy-preserving user discovery to derive strong secrets from low-entropy usernames in an irreversible and unlinkable way. However, it is important to note that it is possible to brute-force search registered usernames, or *crawl* them. This is considered an intrinsic problem of discovery mechanisms [13].

The last contribution of this dissertation is a simulation tool, which functions as a model checker. This tool checks all possible runs of the two protocols within a finite state space to evaluate the two Pudding protocols for completeness and liveness. The evaluation results demonstrate that both Pudding protocols are attainable at the logical level.

One limitation of this study is that it is unable to encompass the entire testing and verification scope for security protocols, as noted in Section 6.4. Therefore, further experimental investigations are needed to estimate the performance of the

Pudding protocols. Performance measurements can be carried out by conducting experiments on real anonymous communication networks or running simulations on appropriate simulation tools such as Mixim [86], Shadow [87], or the Rollercoaster [88] open-source Loopix simulator. On the same note, further research should be undertaken to formally analyse the Pudding protocols. This can be done by modelling the protocols mathematically in a finite state space and using model checkers such as CL-Atse [89], OFMC [90] and SAT-MC [91], or an unbounded space by induction and using model checkers like ProVerif [92], Athena [93], or Scyther [94]. Finally, this study has not evaluated the usability of the protocols because such studies can only be conducted once the protocol-level capabilities are explored and established. A systematic user study to investigate how these features translate to real-life applications is an important next step. One interesting question to answer is how unavailable discovery nodes affect usability.

Although usability and security & privacy have been historically considered as competing notions, this has been changing with the rise of encrypted messaging apps such as Signal, WhatsApp, and Telegram. For the first time, private messaging is accessible to a broad and heterogeneous user base, igniting demand for better privacy from everyday users. A recent example is a widespread backlash fuelled by social media against WhatsApp's change to its privacy policy at the beginning of 2021 [95].

Despite there exists forces against anonymity, such as the Online Safety Bill of the British government [96], I argue that the trend towards increased privacy will eventually result in widespread adoption of anonymous messaging systems. At the time of writing, anonymous messaging apps such as Briar [97] and Jami [98] exist outside the academic literature. These apps also need usable discovery mechanisms. Moreover, it is important to note that although this dissertation has presented these ideas in the context of messaging, the proposed ideas can also allow users to find others to collaborate on a document, to-do list, calendar, etc. in anonymous communication networks.

Bibliography

[1] Alma Whitten and J Doug Tygar. Why Johnny can't encrypt: A usability evaluation of pgp 5.0. In *USENIX Security Symposium*, pages 169–184, 1999.

[2] Signal. `https://signal.org/`, accessed 2021-25-03.

[3] Telegram. `https://telegram.org/`, accessed 2021-12-05.

[4] Whatsapp. `https://www.whatsapp.com/`, accessed 2021-20-04.

[5] General Michael Hayden. The Johns Hopkins foreign affairs symposium presents: The price of privacy: Re-evaluating the NSA, May 2018. `https://www.youtube.com/watch?v=kV2HDM86XgI&t=1070s`, accessed 2021-21-05.

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

[7] Steven J Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *IEEE Symposium on Security and Privacy*, pages 183–195. IEEE, 2005.

[8] Ania M Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The Loopix anonymity system. In *USENIX Security Symposium*, pages 1199–1216, 2017.

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

[10] 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 *Annual Computer Security Applications Conference*, pages 84–99, 2020.

[11] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, 2016.

[12] Joseph Bonneau and Stuart Schechter. Towards reliable storage of 56-bit secrets in human memory. In *USENIX Security Symposium*, pages 607–623, 2014.

[13] Christoph Hagen, Christian Weinert, Christoph Sendner, Alexandra Dmitrienko, and Thomas Schneider. All the numbers are US: Large-scale abuse of contact discovery in mobile messengers. *NDSS. Internet Society*, 2021.

[14] Kun Peng. *Anonymous communication networks: Protecting privacy on the Web.* CRC Press, 2014.

[15] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. *IEEE Symposium on Security and Privacy*, 2003.

[16] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 393–403. Springer, 1995.

[17] Markus Jakobsson, Ari Juels, and Ronald L Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX Security Symposium*, pages 339–353. San Francisco, USA, 2002.

[18] Dan Boneh and Philippe Golle. Almost entirely correct mixing with applications to voting. In *ACM Computer and Communications Security Conference*, pages 68–77, 2002.

[19] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 93–118. Springer, 2001.

[20] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *ACM Computer and Communications Security Conference*, pages 201–210, 2006.

[21] Stevens Le Blond, David Choffnes, William Caldwell, Peter Druschel, and Nicholas Merritt. Herd: A scalable, traffic analysis resistant anonymity net-

work for VoIP systems. In *ACM Conference on Special Interest Group on Data Communication*, pages 639–652, 2015.

[22] Haoyu Zhang, Brian Cho, Ergin Seyfe, Avery Ching, and Michael J Freedman. Riffle: optimized shuffle service for large-scale data analytics. In *EuroSys Conference*, pages 1–15, 2018.

[23] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 302–321. Springer, 2005.

[24] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015.

[25] Michael Harkavy, J Doug Tygar, and Hiroaki Kikuchi. Electronic auctions with private bids. In *USENIX Workshop on Electronic Commerce*, 1998.

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

[27] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. In *Designing Privacy Enhancing Technologies*, pages 96–114. Springer, 2001.

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

[29] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: scaling private contact discovery. *Privacy Enhancing Technologies Symposium*, 2018(4):159–178, 2018.

[30] Moxie Marlinspike. Signal blog - technology preview: Private contact discovery for Signal. `https://perma.cc/G385-EMCB`, accessed 2021-23-05.

[31] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[32] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *USENIX Security Symposium*, pages 991–1008, 2018.

[33] Paul V Mockapetris and Kevin J Dunlap. Development of the Domain Name System. *Computer Communication Review*, 18(4):123–133, 1988.

[34] Sergio Castillo-Perez and Joaquin Garcia-Alfaro. Anonymous resolution of DNS queries. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 987–1000. Springer, 2008.

[35] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends. DNS Security Introduction and Requirements. `https://rfc-editor.org/rfc/rfc4033.txt`, March 2005.

[36] OpenDNS. DNSCrypt. `https://perma.cc/UDW6-7HYA`. accessed on 05/01/2021.

[37] Matthew Dempsky. DNSCurve: Link-level security for the Domain Name System - Internet draft. `https://tools.ietf.org/id/draft-dempsky-dnscurve-00.html`, 2009.

[38] Wouter Wijngaards and Glen Wiley. Confidential DNS - Internet draft. `https://tools.ietf.org/html/draft-wijngaards-dnsop-confidentialdns-03`, 2015. accessed on 07/01/2021.

[39] Phillip Hallam-Baker. Private DNS - Internet draft. `https://tools.ietf.org/html/draft-hallambaker-privatedns-00`, 2014. accessed on 07/01/2021.

[40] Fangming Zhao, Yoshiaki Hori, and Kouichi Sakurai. Analysis of privacy disclosure in DNS query. *International Conference on Multimedia and Ubiquitous Engineering (MUE)*, pages 952–957, 2007.

[41] Rafail Ostrovsky and William E Skeith. A survey of single-database private information retrieval: Techniques and applications. In *International Workshop on Public Key Cryptography*, pages 393–411. Springer, 2007.

[42] Sergio Castillo-Perez and Joaquin Garcia-Alfaro. Evaluation of two privacy-preserving protocols for the DNS. In *2009 Sixth International Conference on Information Technology: New Generations*, pages 411–416. IEEE, 2009.

[43] Liang Zhu, Zi Hu, John Heidemann, Duane Wessels, Allison Mankin, and Nikita Somaiya. Connection-oriented DNS to improve privacy and security. *IEEE Symposium on Security and Privacy*, 2015-July:171–186, 2015.

[44] Paul Schmitt, Anne Edmundson, Allison Mankin, and Nick Feamster. Oblivious DNS: Practical privacy for DNS queries. *Privacy Enhancing Technologies Symposium*, 2019(2):228–244, 2019.

[45] Mahrud Sayrafi. The Cloudflare blog: Introducing DNS resolver for Tor, May 2018. `https://perma.cc/NJY9-GDMD`, accessed 2021-14-05.

[46] Nguyen Phong Hoang, Ivan Lin, Seyedhamed Ghavamnia, and Michalis Polychronakis. K-resolver: towards decentralizing encrypted DNS resolution. *arXiv preprint arXiv:2001.08901*, 2020.

[47] Arjun Nambiar and Matthew Wright. Salsa: a structured approach to large-scale anonymity. In *ACM Computer and Communications Security Conference*, pages 17–26, 2006.

[48] Alan Mislove, Gaurav Oberoi, Ansley Post, Charles Reis, Peter Druschel, and Dan S Wallach. Ap3: Cooperative, decentralized anonymous communication. In *ACM SIGOPS European Workshop*, pages 30–es, 2004.

[49] Andriy Panchenko, Stefan Richter, and Arne Rache. Nisan: network information service for anonymization networks. In *ACM Computer and Communications Security Conference*, pages 141–150, 2009.

[50] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable onion routing with Torsk. *ACM Conference on Computer and Communications Security*, pages 590–599, 2009.

[51] Prateek Mittal and Nikita Borisov. Information leaks in structured peer-to-peer anonymous communication systems. *ACM Transactions on Information and System Security*, 15(1):1–28, 2012.

[52] Qiyan Wang, Prateek Mittal, and Nikita Borisov. In search of an anonymous and secure lookup: attacks on structured peer-to-peer anonymous communication systems. In *ACM Computer and Communications Security Conference*, pages 308–318, 2010.

[53] George Danezis and Paul Syverson. Bridging and fingerprinting: Epistemic attacks on route selection. In *Privacy Enhancing Technologies Symposium*, pages 151–166. Springer, 2008.

[54] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *IEEE Symposium on Security and Privacy*, pages 269–282. IEEE, 2009.

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

[56] George Robert Blakley. Safeguarding cryptographic keys. In *International Workshop on Managing Requirements Knowledge*, pages 313–313. IEEE Computer Society, 1979.

[57] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 72(9):56–64, 1989.

[58] Maurice Mignotte. How to share a secret. In *Workshop on Cryptography*, pages 371–375. Springer, 1982.

[59] Charles Asmuth and John Bloom. A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, 29(2):208–210, 1983.

[60] Josh Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In *Conference on the Theory and Application of Cryptography*, pages 27–35. Springer, 1988.

[61] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Annual Symposium on Foundations of Computer Science*, pages 427–438. IEEE, 1987.

[62] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*, pages 129–140. Springer, 1991.

[63] Elaine Barker, Elaine Barker, William Burr, William Polk, Miles Smid, et al. *Recommendation for key management: Part 1: General.* National Institute of Standards and Technology, Technology Administration, 2006.

[64] Lily Chen. Recommendation for key derivation using pseudorandom functions. *NIST special publication*, 800:108, 2008.

[65] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference*, pages 303–324. Springer, 2005.

[66] Stanislaw Jarecki, Hugo Krawczyk, Maliheh Shirvanian, and Nitesh Saxena. Device-enhanced password protocols with optimal online-offline protection. In *ACM Asia Conference on Computer and Communications Security*, pages 177–188, 2016.

[67] Warwick Ford and Burton S Kaliski. Server-assisted generation of a strong secret from a password. In *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000)*, pages 176–180. IEEE, 2000.

[68] Cameron F Kerry and Patrick D Gallagher. Digital signature standard (DSS). *FIPS PUB*, pages 186–4, 2013.

[69] Tolga Acar, Mira Belenkiy, and Alptekin Küpçü. Single password authentication. *Computer Networks*, 57(13):2597–2614, 2013.

[70] Dave Crocker, Tony Hansen, and Murray Kucherawy. DomainKeys Identified Mail (DKIM) signatures. *ser. RFC6376*, 2011.

[71] Barry Leiba and Jim Fenton. DomainKeys Identified Mail (DKIM): Using digital signatures for domain verification. In *Conference on Email and Anti-Spam*, 2007.

[72] Moxie Marlinspike. Signal blog - the difficulty of private contact discovery. `https://perma.cc/6Y82-QK48`, accessed 2021-08-04.

[73] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *USENIX Security Symposium*, pages 1447–1464, 2019.

[74] Ágnes Kiss, Jian Liu, Thomas Schneider, N Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. *Privacy Enhancing Technologies Symposium*, 2017(4):177–197, 2017.

[75] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *IEEE Symposium on Security and Privacy*, pages 553–567. IEEE, 2012.

[76] Yue Li, Zeyu Chen, Haining Wang, Kun Sun, and Sushil Jajodia. Understanding account recovery in the wild and its security implications. *IEEE Computer Architecture Letters*, (01):1–1, 2020.

[77] Daniele Perito, Claude Castelluccia, Mohamed Ali Kaafar, and Pere Manils. How unique and traceable are usernames? In *Privacy Enhancing Technologies Symposium*, pages 1–17. Springer, 2011.

[78] Yongjun Li, You Peng, Zhen Zhang, Hongzhi Yin, and Quanqing Xu. Matching user accounts across social networks based on username and display name. *World Wide Web*, 22(3):1075–1097, 2019.

[79] Reza Zafarani and Huan Liu. Connecting corresponding identities across communities. In *International AAAI Conference on Web and Social Media*, volume 3, 2009.

[80] Reza Zafarani and Huan Liu. Connecting users across social media sites: a behavioral-modeling approach. In *19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 41–49, 2013.

[81] Joseph Bonneau, Elie Bursztein, Ilan Caron, Rob Jackson, and Mike Williamson. Secrets, lies, and account recovery: Lessons from the use of personal knowledge questions at google. In *24th International Conference on World Wide Web*, pages 141–150, 2015.

[82] Stuart Schechter, AJ Bernheim Brush, and Serge Egelman. It's no secret. measuring the security and reliability of authentication via "secret" questions. In *IEEE Symposium on Security and Privacy*, pages 375–390. IEEE, 2009.

[83] Wendy Goucher. Email passwords: pushing on a latched door. *Computer Fraud & Security*, 2012(9):16–19, 2012.

[84] Facebook. Facebook help centre: How can I choose friends to help me log in if I ever get locked out of my Facebook account? `https://perma.cc/NN2X-37D3`, accessed 2021-09-05.

[85] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking.* MIT press, 2018.

[86] Iness Ben Guirat, Devashish Gosain, and Claudia Diaz. Mixim: A general purpose simulator for mixnet. *Privacy Enhancing Technologies Symposium - HotPETs Workshop*, 2020.

[87] Rob Jansen and Nicholas Hopper. Shadow: Running tor in a box for accurate and efficient experimentation. In *Symposium on Network and Distributed System Security (NDSS).* Internet Society, February 2012.

[88] Daniel Hugenroth, Martin Kleppmann, and Alastair Beresford. Rollercoaster : An Efficient Group-Multicast Scheme for Mix Networks. *In submission to USENIX Security*, 2021.

[89] Mathieu Turuani. The CL-Atse protocol analyser. In *International Conference on Rewriting Techniques and Applications*, pages 277–286. Springer, 2006.

[90] David Basin, Sebastian Mödersheim, and Luca Vigano. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[91] Alessandro Armando and Luca Compagna. SATMC: a SAT-based model checker for security protocols. In *European Workshop on Logics in Artificial Intelligence*, pages 730–733. Springer, 2004.

[92] Bruno Blanchet et al. An efficient cryptographic protocol verifier based on prolog rules. In *CSFW*, volume 1, pages 82–96. Citeseer, 2001.

[93] Dawn Xiaodong Song. Athena: a new efficient automatic checker for security protocol analysis. In *IEEE Computer Security Foundations Workshop*, pages 192–202. IEEE, 1999.

[94] Cas JF Cremers. The Scyther tool: Verification, falsification, and analysis of security protocols. In *International Conference on Computer Aided Verification*, pages 414–418. Springer, 2008.

[95] Mike Isaac. WhatsApp delays privacy changes amid user backlash, January 15 2021. `https://perma.cc/EZ2N-GL7X`, accessed 2021-25-05.

[96] UK Department for Digital, Culture, Media & Sport. Draft Online Safety Bill, May 12 2021. `https://www.gov.uk/government/publications/draft-online-safety-bill`, accessed 2021-02-06.

[97] Briar Project. `https://briarproject.org/`, accessed 2021-14-03.

[98] Jami. `https://jami.net/help/`, accessed 2021-14-03.