KTH Computer Science
and Communication

Privacy Issues in Decentralized Online Social Networks and other Decentralized Systems

BENJAMIN GRESCHBACH

Doctoral Thesis
Stockholm, Sweden 2016

TRITA-CSC-A 2016:28
ISSN 1653-5723
ISRN KTH/CSC/A--2016/28-SE
ISBN 978-91-7729-194-7

KTH
School of Computer Science
and Communication
SE-100 44 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framläges till offentlig granskning för avläggande av teknologie doktorsexamen i datalogi 15/12-2016 klockan 14.00 i sal F3, Kungl Tekniska högskolan, Stockholm.

Tryck: Universitetsservice US AB

Abstract

Popular Online Social Networks (OSNs), such as Facebook or Twitter, are logically centralized systems. The massive information aggregation of sensitive personal data at the central providers of these services is an inherent threat to the privacy of the users. Leakages of these data collections happen regularly – both intentionally, for example by selling of user data to third parties and unintentionally, for example when outsiders successfully attack a provider.

Motivated by this insight, the concept of Decentralized Online Social Networks (DOSNs) has emerged. In these proposed systems, no single, central provider keeps a data collection of all users. Instead, the data is spread out across multiple servers or is distributed completely among user devices that form a peer-to-peer (P2P) network. Encryption is used to enforce access rights of shared content and communication partners ideally connect directly to each other. DOSNs solve one of the biggest privacy concerns of centralized OSNs in a quite forthright way – by getting rid of the central provider. Furthermore, these decentralized systems can be designed to be more immune to censorship than centralized services. But when decentralizing OSNs, two main challenges have to be met: to provide user privacy under a significantly different threat model, and to implement equal usability and functionality without centralized components.

In this work we analyze the general privacy-problems in DOSNs, especially those arising from the more exposed metadata in these systems. Furthermore, we suggest three privacy-preserving implementations of standard OSN features, i. e., user authentication via password-login, user search via a knowledge threshold and an event invitation system with fine-grained privacy-settings. These implementations do not rely on a trusted, central provider and are therefore applicable in a DOSN scenario but can be applied in other P2P or low-trust environments as well. Finally, we analyze a concrete attack on a specific decentralized system, the Tor anonymization network, and suggest improvements for mitigating the identified threats.

Sammanfattning

Populära sociala nätverkstjänster som Facebook och Instagram bygger på en logiskt centraliserad systemarkitektur. Tjänsteleverantörerna har därför tillgång till omfattande ansamlingar av känsliga personuppgifter, vilket innebär en oundviklig risk för integritetskränkningar. Med jämna mellanrum läcks dessa informationsansamlingar till tredje part – antingen när tjänsteleverantören själv säljer eller ger dem till externa aktörer, eller när obehöriga får åtkomst till tjänsteleverantörens datasystem.

Decentraliserade sociala nätverkstjänster (*eng.* Decentralized Online Social Networks, DOSNs) är en lovande utveckling för att minska denna risk och för att skydda användarnas personliga information såväl från tjänsteleverantören som från tredje part. Ett vanligt sätt att implementera ett DOSN är genom en icke-hierarkisk nätverksarkitektur (*eng.* peer-to-peer network) för att undvika att känsliga personuppgifter samlas på ett ställe som är under tjänsteleverantörens kontroll. Kryptering används för att skydda kommunikationen och för att realisera åtkomstkontrollen av information som ska delas med andra användare.

Att inte längre ha en tjänsteleverantör som har tillgång till all data innebär att den största riskfaktorn for integritetskränkningar tas bort. Men genom att ersätta den centrala tjänsteleverantören med ett decentraliserat system tar vi även bort ett visst integritetsskydd. Integritetsskyddet var en konsekvens av att förmedlingen av all användarkommunikation skedde genom tjänsteleverantörens servrar. När ansvaret för lagring av innehållet, hantering av behörigheterna, åtkomst och andra administrativa uppgifter övergår till användarna själva, blir det en utmaning att skydda metadata för objekt och informationsflöden, även om innehållet är krypterat. I ett centraliserat system är dessa metadata faktiskt skyddade av tjänsteleverantören – avsiktligt eller som en sidoeffekt.

För att implementera de olika funktioner som ska finnas i ett integritetsskyddande DOSN, är det nödvändigt både att lösa dessa generella utmaningar och att hantera frånvaron av en betrodd tjänsteleverantör som har full tillgång till all data. Användarautentiseringen borde till exempel ha samma användbarhet som i centraliserade system. Det vill säga att det är lätt att ändra lösenordet, upphäva rättigheterna för en stulen klientenhet eller återställa ett glömt lösenord med hjälp av e-post eller säkerhetsfrågor – allt utan att förlita sig på en betrodd tredje part. Ett annat exempel är funktionen att kunna söka efter andra användare. Utmaningen där är att skydda användarinformationen samtidigt som det måste vara möjligt att hitta användare baserad på just denna informationen. En implementation av en sådan funktion i ett DOSN måste klara sig utan en betrodd tjänsteleverantör som med tillgång till alla användardata kan upprätthålla ett globalt sökindex.

I den här avhandlingen analyserar vi de generella risker för integritetskränkningar som finns i DOSN, särskilt de som orsakas av metadata. Därutöver föreslår vi tre integritetsskyddande implementationer av vanliga funktioner i en social nätverkstjänst: lösenordsbaserad användarautentisering, en användarsökfunktion med en kunskapströskel och en inbjudningsfunktion för

evenemang med detaljerade sekretessinställningar. Alla tre implementationerna är lämpliga för DOSN-scenarier eftersom de klarar sig helt utan en betrodd, central tjänsteleverantör, och kan därför även användas i andra sammanhang såsom icke-hierarkiska nätverk eller andra system som måste klara sig utan en betrodd tredje part. Slutligen analyserar vi en attack på ett specifikt decentraliserat system, anonymitetstjänsten Tor, och diskuterar hur systemet kan skyddas mot de analyserade sårbarheterna.

Acknowledgements

First of all, I want to thank my supervisor Sonja Buchegger for all the support I got during the last years. I am very grateful for all the profound knowledge she shared, both of the research topics and all the essential methodology skills a PhD student needs. She is extraordinary good at creating a positive and constructive working environment – when things don't go as they should, she's asking the important questions, and her reliable humor is able to lift the mood of any group meeting. With her academic experience and deep knowledge of the subjects she always gave excellent feedback and helped me to find the answer to any question I had.

I am also grateful for the support and inspiring discussions with my former and current fellow group members Gunnar Kreitz, Oleksandr Bodriagov, Guillermo Rodríguez Cano, and Daniel Bosk. And I want to thank the internal reviewer Mats Näslund who gave valuable feedback on the thesis draft.

Furthermore, I want to thank all people I met at TCS and KTH, especially Adam Schill Collberg, Andreas Lindner, Andreas Lundblad, Andreas Selamtzis, Björn Terelius, Cenny Wenner, Christoph Baumann, Dilian Gurov, Emma Enström, Emma Riese, Fei Chen, Fei Niu, Freyr Sævarsson, Grzegorz Milka, Gurvan Le Guernic, Hamed Nemati, Hojat Khosrowjerdi, Ilario Bonacina, Jan Elffers, Jana Götze, Jesús Giráldez Crú, Jonas Haglund, Joseph Swernofsky, José David Lopes, Karl Palmskog, Katrin Titma, Linda Kann, Lukáš Poláček, I lost the game again, Marc Vinyals, Mateus de Oliveira Oliveira, Mika Cohen, Mladen Mikša, Muddassar Azam Sindhu, Musard Balliu, Narges Khakpour, Oliver Schwarz, Pedro de Carvalho Gomes, Per Austrin, Rajsekar Manokaran, Raveesh Meena, Richard Glassey, Roberto Guanciale, Roelof Pieters, Sangxia Huang, Siavash Soleimanifard, Stefan Nilsson, Susanna Figueiredo de Rezende, Thatchaphol Saranurak, Tobias Mömke, Torbjörn Granlund, Vahid Mosavat and Xin Zhao for lunch company, Farsi lessons, failed kite-kayaking attempts, modern circular chess games and trips to the archipelago.

Finally I want thank my friends and family, especially my parents for their invaluable support. They all know that they mean much more to me than what I could write about here.

Contents

1

1. Introduction

The service providers of today's popular Online Social Network (OSN) services, such as Facebook, Twitter or Instagram, are driven by business models largely based on targeted advertisement. These companies count hundreds of millions of users of their services and have a vital interest in collecting as much information about them as possible. And the possibilities are manifold. Service providers can not only analyze all content that users actively publish or share with friends but also leverage indirectly disclosed information such as online times of a user and correlate information about one user with data they hold about all other users. This massive aggregation of personal data of millions of users at very few, logically centralized actors is seen as a fundamental threat to user privacy by many civil rights and privacy advocates[1]. These concerns are fueled by the regular leakages of centrally collected user data caused by security breaches[2] or the deliberate distribution of user data to third parties such as advertising companies[3] or secret services[4].

One approach to mitigate this inherent privacy issue of logically centralized OSN services is decentralization. The basic idea of so called Decentralized Online Social Networks (DOSNs) is to provide social network service functionalities without the need of any central, trusted party. This is realized either by distributing the service to multiple servers or by completely decentralizing the system, building on a peer-

[1] For example the Electronic Frontier Foundation (EFF) `https://www.eff.org/issues/social-networks`, retrieved 2016-11-01, or Privacy International `https://www.privacyinternational.org/node/8`, retrieved 2016-11-01.

[2] For example Twitter leaking data from 250 thousand users in February 2013 (`http://blog.twitter.com/2013/02/keeping-our-users-secure.html`, retrieved 2016-11-01) or the theft of 500 million Yahoo user profiles in late 2014 (`https://yahoo.tumblr.com/post/150781911849/an-important-message-about-yahoo-user-security`, retrieved 2016-11-01).

[3] For example Facebook selling user data (`http://www.telegraph.co.uk/technology/facebook/8917836/Facebook-faces-EU-curbs-on-selling-users-interests-to-advertisers.html`, retrieved 2016-11-01). Some consider this an inherent consequence of the providers' business model and point out that for these providers, "people are not customers, but primarily products" [65].

[4] As for example under the "PRISM" program that allows U.S. intelligence services access to the user data of at least nine Internet companies (`http://www.washingtonpost.com/wp-srv/special/politics/prism-collection-documents/`, retrieved 2016-11-01).

to-peer (P2P) network formed by the user devices themselves. Cryptography is commonly used in these systems to encrypt communication and enforce access rights on content. Thus DOSNs avoid the aggregation of sensitive user data at locations controlled by a single provider. The decentralization approach can even support the scalability of the network – each new user does not only consume but also contributes potential new communication and storage resources – and facilitate censorship-resistance as shutting down a couple of servers is not enough to bring the system down [4, 12]. So recruiting the participants for data management and communication might not only be beneficial from the privacy perspective but also advantageous for the scalability and resilience of a system.

While the absence of a single point of data aggregation solves one of the most important privacy problems, the decentralization also removes some privacy protection afforded by the provider's intermediation of all communication in a centralized OSN. As content storage, access right management, retrieval and other administrative tasks of the service become the obligation of the users, it is non-trivial to hide the metadata of objects and information flows, even when the content itself is encrypted. In a centralized system the provider is protecting such metadata, deliberately or as a side effect, from external adversaries.

So when decentralizing OSNs, two main challenges have to be met: To provide user privacy under this significantly different threat model. And to implement equal usability and functionality without centralized components.

An example for the first challenge – to ensure user privacy in a decentralized system – is the problem of inferences from ciphertexts. Even when content is encrypted only for the intended recipients, the size or structure of the ciphertext may reveal information such as the number of elements in a list or even serve as an invariant fingerprint of a specific content that can be used to track how it spreads throughout the network. In Article A we give a systematic overview of these threats arising from metadata generated in a DOSN and list possible countermeasures.

An example for the second challenge – to implement equal functionality without central components – is the difficulty to implement a password-reset functionality for users who forgot their password in a system without a trusted, central provider: In a centralized system, the provider can send a new password to an e-mail address or a phone number that the user provided earlier. In a decentralized system, no single involved party can be trusted with this sensitive task, because everyone endowed with that power will necessarily be able to access and compromise all user data. In Article B we present a protocol that aims at implementing a similar functionality and other password management tasks in a decentralized way. Another example for the challenge of implementing OSN functionality without a central provider is user search. There, the challenge is to protect user data while making users findable at the same time. An implementation of such a feature in a DOSN has to work without assuming a trusted provider having access to all user profiles maintaining a global search index. In Article C we suggest an implementation of such a decentralized user search.

This thesis aims at mapping out these new challenges to user privacy in the

context of Decentralized Online Social Networks. It contributes to the DOSN research with suggestions how to meet some of these challenges, suggests distributed and privacy-preserving protocols for three specific OSN features and analyzes one concrete attack on a deployed decentralized system.

2

2. Privacy

Privacy is a complex concept touching many disciplines, such as philosophy, sociology, anthropology, law, politics and technology. It has a long history and its roots can be traced back as far as to Aristotele's distinction between the public and private sphere [42]. In 1890, Warren and Brandeis published an influential law review article [138], advocating for a "right to privacy" that by then was not yet recognized as an explicit right on its own in the US. Their main concern was the publication of details and photographs of the private lives of people, for example in newspapers. Thus they conceptualized privacy as "the right to be let alone", i. e., the right to social retreat. Another influential work on the concept of privacy was published in 1967 by Westin [140], tracing privacy back to "a tradition of limiting the surveillance powers of authorities over the private activities of individuals and groups", and framing it in terms of informational self-determination, i. e., the "claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others." More recent conceptualizations of privacy view it as "control over an aspect of the identity one projects to the world" and the "freedom from unreasonable constraints on the construction of one's own identity" [5], or a right to contextual integrity [98], "demanding that information gathering and dissemination be appropriate to that [specific] context and obey the governing norms of distribution within it", thus preventing unwanted information flow from one context to another.

Privacy in Online Social Networks

In the context of OSNs, Gross et al. [64] have identified several privacy threats such as stalking, de-anonymization of medical records, identity theft, e. g., by social insurance number reconstruction, user profiling and simplified social engineering. Danezis et al. [40] pointed out that the position of a user in a social network reveals characteristics about the person, without being stated explicitly in the published content, such as their status and potential influence reach. Paul et al. [102] underline the consequences of massive central data aggregation in conjunction with an advertising-based business model of major OSN providers. They warn against the

risks of direct misuse or unintended leakage of this data that is not appropriately protected and hard to anonymize.

Krishnamurthy and Wills [84] show how personally identifiable information was leaked from different OSN services to third party servers via HTTP header information and tracking cookies in 2010. Facebook, currently the OSN with the largest number of active users[1], was criticized for leaking personal information and preferences with third party sites[2], misleading people into accidentally sharing private information with unintended audience due to confusing privacy settings[3], tracking browsing behaviour of users even when logged out or without explicit consent[4] and not complying with privacy regulations[5].

The Different Tales of Privacy

Gürses and Diaz [66] make an important observation about the different conceptions of privacy that are used in OSN privacy research. They distinguish three types of problem formulations: privacy as *surveillance* problem, *social* privacy and *institutional* privacy. The different types make different assumptions, use different definitions and methods, and differ in the suggested solutions. At the same time they are obviously entangled.

The surveillance problem perspective, found often in Privacy Enhancing Technologys (PETs) research, focuses on protecting users from confidentiality breaches or Denial of Service (DoS) attacks, with the typical example of activists engaged in political dissent. Except for the intended recipients of shared information, all other entities in the OSN context, including the OSN provider and state actors, are considered to be adversarial. So any explicit or implicit data disclosure is tried to be prevented. These approaches also often try to empower users to circumvent censorship.

The social privacy approach considers the OSN provider to be a trusted entity. It focuses on enabling the users to choose the correct audience when sharing content in order to prevent embarrassing or regrettable sharing of information with unintended recipients. So this perspective focuses on social boundary negotiations, the semantics of and the context in which published data appears, but pays less attention to implicit data disclosures such as generated behavioral data observable by a provider. A common method of this type of approach is to increase the

[1] `http://www.ebizmba.com/articles/social-networking-websites`, retrieved 2016-11-01

[2] `https://www.eff.org/deeplinks/2010/04/handy-facebook-english-translator`, retrieved 2016-11-01

[3] `http://www.cbsnews.com/news/zuckerberg-family-pic-stirs-facebook-privacy-debate/`, retrieved 2016-11-01

[4] `http://www.usatoday.com/tech/hotsites/2009-09-21-facebook-beacon_N.htm`, retrieved 2016-11-01

[5] `http://www.irishexaminer.com/ireland/facebook-wont-like-its-17th-complaint-165606.html`, retrieved 2016-11-01

users' awareness and control over the possible and actual spreading of the shared information.

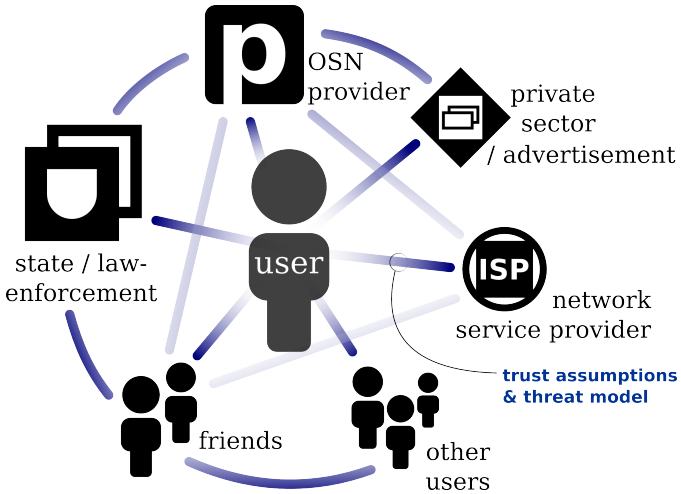


Figure 2.1: A specific privacy perspective makes explicit or implicit assumptions about the trust relations between the actors involved in the OSN context.

The institutional privacy perspective does, like the surveillance perspective, not trust the OSN provider, but considers, unlike the surveillance perspective, state institutions as trusted control instances, being able to exercise regulatory power over providers and other private sector actors involved in the management of personal data. Thus it can be found mainly in research on technical solutions for compliance and accountability with respect to privacy laws or policies, regulating data collection and processing for organizations.

Privacy Perspective of this Thesis

The work presented in this thesis takes mainly the privacy as surveillance problem perspective. It is concerned about the extent of state level actors' surveillance practices, which has been speculated about already before, but got confirmed in a worrying comprehensive way by the revelations of the former NSA contractor Edward Snowden in 2013. The power of secret services that can observe virtually every citizen's online activities in real-time when targeted and the possibility of automatically analyzing a large fraction of all Internet traffic is seen as a threat to the development of free and democratic societies. Companies that operate popular OSN services, such as Google and Facebook, are mentioned as being included in the "PRISM" program, allowing secret service staff to search all available data of arbi-

trary users of these services by collecting it directly from the companies' servers.[6] This is one motivation for not trusting central OSN providers. Approaches where the provider only handles end-to-end encrypted data of users and therefore does not need to be trusted have the problem of still centrally collecting metadata, e. g., inferred social graphs from analyzing which users access which other users' data. These providers might also be priority targets of censorship when state actors realize that they cannot access the user data anymore. Therefore, the work presented in this thesis is focusing on decentralization solutions to achieve more comprehensive user privacy in OSNs and other distributed systems.

Even though having the main focus on the surveillance perspective, some of the included work also takes the social privacy perspective. When in the decentralized OSNs a friend's device might take over data handling that before had been done by the central provider, new social adversary models emerge and touch the questions central to the social privacy perspective such as social boundary negotiation and leakage of data to unintended recipients.

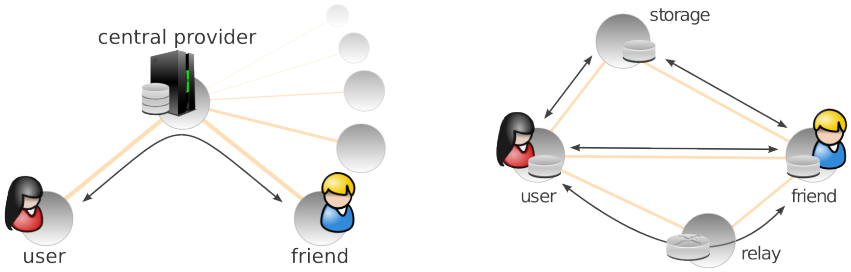
[6] `http://www.washingtonpost.com/wp-srv/special/politics/prism-collection-documents/`, retrieved 2016-11-01

3. Decentralized Online Social Networks (DOSN)

3

In a centralized OSN the users must trust the provider to enforce access rights correctly, not to leak or misuse the content provided by the users, and to be sufficiently secured against third-party attacks. Furthermore, the interests of the users might not be aligned with those of the provider, for example users want their data to be available even if it is not attractive for advertising any longer [118].

To move away from a system design with a central point of massive personal data aggregation and to put the users back into control of their own data, the decentralization of OSNs has been proposed already about ten years ago. Since then, DOSNs have become the focus of a lively branch of research (for early work on the topic see for example [10, 14, 26, 27, 39, 144], for surveys [35, 102, 103]).



(a) All communication is relayed by the central provider.

(b) Besides direct peer communication, several other nodes can be involved.

Figure 3.1: Logical communication flows in a) centralized and b) decentralized OSN architectures.

There are many different proposals how to decentralize an OSN, but what they have in common is the aim of making the system independent of a single, central provider as illustrated in Figure 3.1. There is a wide range of designs, spanning from more to less decentralized network architectures. In the completely decentralized approaches, the users themselves form a peer-to-peer (P2P) network in order to

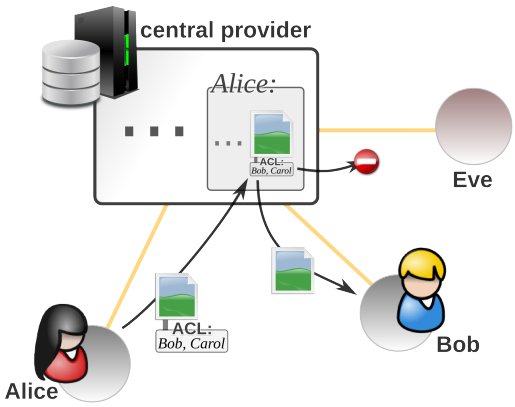


Figure 3.2: Access right enforcement in a centralized OSN: The provider is trusted both with the plaintext content and to correctly enforce the access rights that the author defined.

collaboratively provide the storage and communication infrastructure for the OSN service. Other proposals have a distributed servers model, keeping the client-server paradigm but giving users more freedom to choose which server they want to trust or even run their own server. Diaspora[1] and Friendica[2] are deployed examples of this distributed servers model. The work in this thesis and the subsequent discussion of DOSNs focuses, however, on approaches that are more on the completely decentralized side of the scale, i. e., DOSNs with no or as few as possible centralized components. For these approaches users neither need to trust any server nor have to have the technical equipment and knowledge to run an own server.

Using encryption as access control mechanism and a P2P network as underlying communication structure was already part of the first proposals for DOSN implementations [27]. In these proposals, access control for published content is enforced by data encryption. So instead of having one party that is trusted with the plaintext data and authenticates requests from other users (Figure 3.2), the data itself is encrypted already before it is shared with anyone. The encrypted content can only be decrypted by the recipients that the author intended to share the content with. Therefore, it becomes less crucial where the data is stored and less trust is required for storage nodes (Figure 3.3).

Furthermore, users keep the physical and legal ownership of their content to a larger extent. They might store replicas on other nodes in the network to increase redundancy and availability but also host their own data. This facilitates data portability and can be beneficial for increasing censorship-resistance and resilience

[1] `https://github.com/diaspora/diaspora/`, retrieved 2016-11-01
[2] `https://github.com/friendica/friendica/`, retrieved 2016-11-01

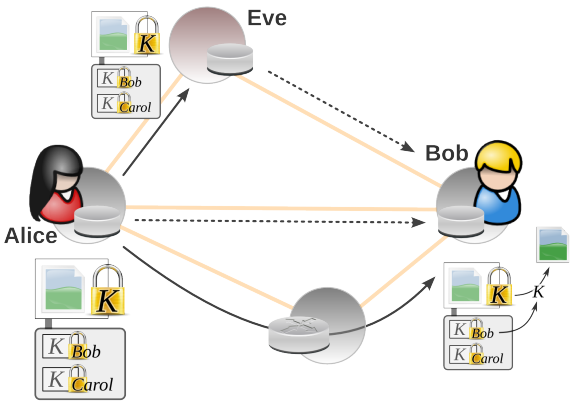


Figure 3.3: Access right enforcement in a DOSN: The author encrypts the content for the intended recipients, so less trusted nodes can be used to store the data. In this example, Alice encrypts the content with a symmetric key K and encrypts K with the public keys of the intended recipients Bob and Carol.

with respect to network outages. In a centralized system it might be enough to shut down a small number of important servers to take the whole system down while in a DOSN the failure of a few servers will likely only effect a small part of the network.

In the following I will discuss some basic components and aspects of DOSNs in more detail to describe the context in which the research articles of this thesis are located. This is neither a comprehensive survey of proposed DOSN systems nor a complete overview of all DOSN components and functionality, but it aims to give a brief overview of how distributed storage, content encryption, network communication, identity management and social relationship information management can be implemented in DOSN systems and what challenges they face. Finally, I will have a brief look at deployed systems and discuss their properties.

Distributed Storage

The requirements for a distributed storage that is suitable for a DOSN are challenging: in contrast to P2P file sharing applications it needs to be able to store a high number of unpopular objects, for example like-indications or comments that are only relevant for very few users, and, in contrast to for example P2P backup systems, handle frequent read and write requests for these objects [103].

Distributed Hash Tables (DHTs) are key–value storage systems where the responsibility of storing a value for a given key is assigned to one or several out of a usually high number of participating users. Furthermore, a DHT has a routing mechanism, that allows all participating users to find the user who is responsible for

storing the value for a certain key. As most DHT implementations do not rely on any central component, they have been suggested to serve as a distributed storage for DOSNs [6]. They can either be used to store smaller data objects directly or to hold an index, containing pointers to the actual storage locations of possibly larger content objects. DHTs face different challenges such as achieving high availability even when old users are leaving and new users are joining the system (churn), guaranteeing direct connectivity between arbitrary users even when they are located behind firewalls or Network Address Translation (NAT) boxes, and Sibyl attacks [113]. Sibyl attacks, i. e., attacks where an adversary pollutes a system with a large number of fake users under her control, can be used to monitor requests in a DHT or to manipulate the stored content, e. g., to serve a different value than the one that was stored under a certain key, or to effectively delete objects [133]. When being deployed in conjunction with a DOSN, the social relationship information from the DOSN can be used to protect the DHT from Sibyl attacks [145] or to inform the DHT routing mechanism to increase performance [96].

While general DHT approaches store data at random nodes in the network, some authors suggest to store profile data at trusted friends, where trust is defined as having chosen a privacy policy that allows this friend to view all data [117]. Other suggested approaches, such as the DOSN "Safebook" [39], use trust relations not only for storage placement strategies but also for routing to increase communication anonymity. To increase availability of the DOSN system, some authors propose hybrid solutions for the decentralization–centralization dimension, i. e., having highly available "super peers" responsible for crucial tasks such as bootstrapping new users, acting as a backup storage system while no other nodes are available to replicate a user's data [120].

Content Encryption

The choice of suitable cryptographic ciphers for data encryption in DOSNs is another branch of research. It deals with finding efficient encryption methods for the context of OSNs, where many small objects, such as like indications or small text snippets, continuously need to be encrypted to varying sets of recipients. Efficiency can be gained by leveraging the social structure of the communication parties, e. g., having group keys for frequently used audience sets. At the same time, encryption headers, ciphertext structures and specific communication patterns that result from the chosen encryption method should not give away sensitive information such as the number of recipients or the type of the encrypted object. Attribute-based encryption schemes [14] or broadcast encryption schemes [22] have been studied for these purposes.

Network Communication Strategies

Related to both the storage implementation and the choice of cryptographic ciphers for content encryption is the design of the network communication. The strategies for when to send, how to route, how to cache and where to store information can be different for the different components of a DOSN, such as direct messaging, group messaging, chats, search or file-sharing features. Unstructured routing strategies such as flooding (recursively sending to all known peers) or gossip communication (recursively sending to a subset of peers) can for example be used for disseminating search queries in the absence of a global search index. Structured approaches such as DHT-based recipient address lookup can for example be used to locate the responsible storage node for receiving asynchronous messages on behalf of a user that is offline.

A common way to present the most recent publishing activities of a user's friends is the friend activity feed, also called "newsfeed" or "timeline". This is an example for an OSN feature that involves a network communication design choice: the problem of how to distribute published user content, e. g., wall-posts, comments, or pictures. On a high abstraction level, there are two different strategies: the user who renders the friend activity feed can fetch the relevant information from all friends (pull strategy), or these updates can be sent out by the users who publish the content to all intended recipients (push strategy). Both strategies can have many different possible implementations and strategies that mix both push and pull aspects are possible as well, as shown in the example illustrated in Figure 3.4.

In this example, rendering time of a friend activity feed is reduced by pushing small preview information objects (e. g., a thumbnail of a larger image together with information where to retrieve the full image) to dedicated user inboxes. These inboxes are made available by the distributed storage even when the user is offline. Users can subscribe to different walls to specify where to get push-notifications from. When logging in after having been offline for some time, a user's client will display all push notifications found in the user's inbox first and then subsequently pull the complete entries and entries that might not have been pushed to the user inbox from all walls the user subscribed to. Access control can be added to the push-queue to increase the privacy of the subscribers. In the above example, user A could for example store the subscription information in the push-queue of W_C only in encrypted form, so that only A's friends learn that A subscribed to W_C. When these friends publish content on that wall, they can decrypt the subscription information and send push notifications to A's inbox. This comes with the trade-off that users that are not friends with A, but also publish on W_C (for example because they are friends with C but not with A), will not be able to send push notifications to A. As the push-mechanism is only used for performance improvements, A will eventually pull all content published on W_C, but posts from direct friends will be rendered first – which might be a reasonable priority policy for which entries are relevant and to be displayed first.

The above example is an illustration of a solution using a structured overlay,

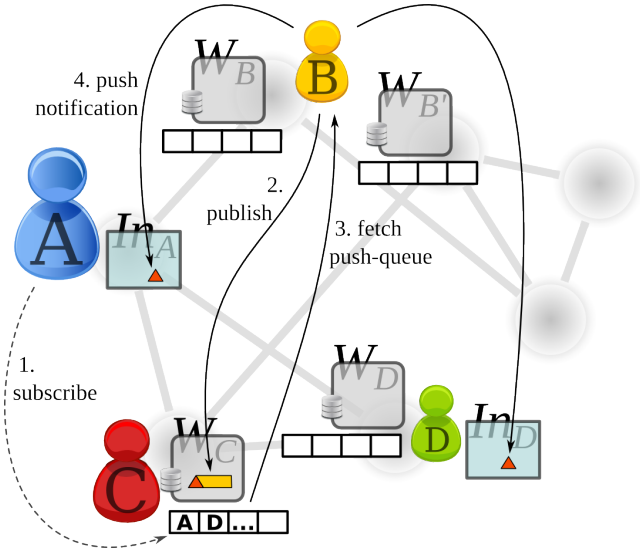


Figure 3.4: Example for a conditional push-model: User A subscribes to user C's wall W_C (1). When B publishes content on that wall (2), B can fetch the information of subscribed users (3) and push a notification to user A's inbox In_A, user D's inbox In_D, and so on (4).

i. e., a DHT or another distributed storage where direct addressing of user walls and inboxes is possible. Some authors have discussed solutions to disseminate user profile updates that work even in unstructured friend-to-friend networks, where communication is only possible between directly connected friends. These networks have widely non-uniform clustering characteristics which poses challenges for classical gossip protocols, but these protocols can be adapted to work efficiently also in these situations [92].

Identity Management

Without a central provider, account creation cannot be implemented by the classical procedure of a user registering with a service provider. To come up with suitable alternatives in a decentralized system is one of the challenges for a DOSN identity management system. It includes granting registered users access to the right resources at a later point in time, to allow them to access these resources from different devices and to guarantee the uniqueness of user identifiers, so that one username, for example, cannot be registered by more than one user.

For many distributed systems the identity of a user is simply a cryptographic key. Usually, a keypair is generated at setup and the public key becomes the user's permanent identifier that is used to recognize the user after ephemeral session

credentials have expired, for example, the next time the user connects to the system. While this solution works well for many systems and has the advantage of offering good security as long as reasonable key-lengths are chosen, it is not enough for systems that feature social interactions and therefore require identifiers that can be chosen by the user and that are more readable and memorable by humans than a cryptographic key fingerprint. Public keys also become impractical as identifiers for systems that allow users to log in from different devices, because there is no straightforward method for manually transferring cryptographic key material to a new device that is both easy to use and secure.

One partial solution can be to bind human-friendly usernames to cryptographic keys in a secure way. The twister application [57] demonstrates how this can be done in a completely decentralized system by leveraging the block-chain technology that the crypto currency Bitcoin [95] is based on: the block-chain, a history of events strictly sorted by time and with a consistent global view available to all users, can be used to record mappings of usernames to public keys. Uniqueness is guaranteed by a first-come–first-served policy that only considers the very first registration of a username to be valid and that can be verified by all users because of the registration history being public and non-malleable.

The problem of how to transfer credentials in a usable and secure way from one device to another is one of the concerns of Article B contained in this thesis. The proposed solution emulates the traditional username/password-login paradigm in a decentralized system.

Social Relationship Information

Related to identity management is the management of social relationship information, e. g., the list of "friends" of a user. Social relationship information can be used for access control, data presentation, user-index structuring or relation announcements. For access control, users can use it for example to only allow friends, a certain subset of them or friends of friends to access certain resources. Data presentation policies can make use of it to show content from close friends more prominently in friend activity feeds. By browsing the list of friends of a friend, users get an implicitly structured view of the user-index of a social network. And finally, users might want to share which friendships they have established with other users for example by making a list of friends available to other users or announce these relations in other ways.

There are several privacy–functionality trade-offs involved when implementing social relationship information management. For example, when using social relationship information for access control, the cryptographic headers of encrypted objects might leak some information about social relations of a user, as mentioned in the description of content encryption above.

Another research question is how to implement more complex social-graph based access control policies, such as sharing content with friends-of-a-friend. The prob-

lem with these policies is that they might clash with the privacy interest of other users – if a friend of mine does not want to share her list of friends with me, how can I share content with these friends of my friend? Backes et al. [13] designed identity management and authentication protocols based on zero-knowledge proofs that allow such complex social-graph based policies without compromising user privacy. Figure 3.5 illustrates an example: if user A is friend with user B, the protocol allows a user C, who is friend with B to proof anyone (including A), that she is friend of a friend of A in an anonymous manner, that is, the proof does neither give away C's identity nor that B is the common, connecting link.

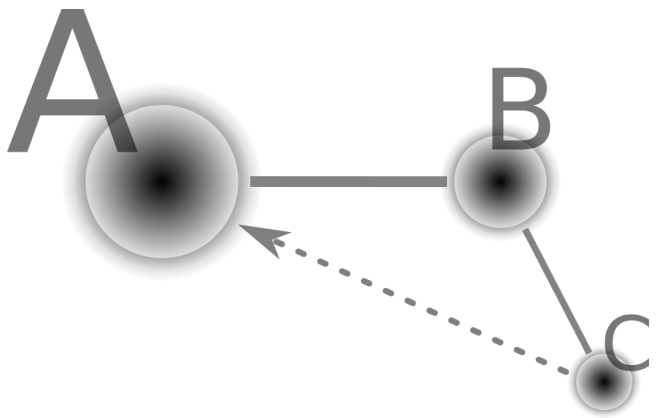


Figure 3.5: User A and B are friends and user B and C are friends (solid lines). User C is not a friend of A but a friend of a friend (dashed arrow).

Other authors have proposed to use a decentralized version of anonymous credentials to achieve similar goals [60].

Deployed Systems

There are a couple of deployed systems that fall into the category of partly decentralized OSNs, following a distributed server model. Among those are Diaspora[3], Friendica[4], pump.io[5] and GNU Social[6], with Diaspora being more famous due to mainstream media attention in 2010, when the initial developer team raised $ 200,000 in a crowdfunding campaign, 20 times more than aimed for[7]. Today, Diaspora, has at least 60,000 active users according to self-reported statistics from servers that publish these statistics[8]. These systems consist of many distributed

[3] `https://github.com/diaspora/diaspora/`, retrieved 2016-11-01
[4] `https://github.com/friendica/friendica/`, retrieved 2016-11-01
[5] `https://github.com/pump-io/pump.io`, retrieved 2016-11-01
[6] formerly known as StatusNet, `https://gnu.io/social/`, retrieved 2016-11-01
[7] `http://nyti.ms/1ROpxpi`, retrieved 2016-11-01
[8] aggregated at `https://the-federation.info/`, retrieved 2016-11-01

servers (called pods in Diaspora) that connect to each other. Each server can host several user accounts that can be accessed via a web-interface that the server provides. Users can either operate an own server or choose an existing one to register a user account. In contrast to centralized systems there is not one single provider, but users have the freedom to choose their provider or become their own. Both choices entail, however, significant drawbacks. If users choose to register on an existing server, they need to trust the operator of the server who either stores user data unencrypted or can gain access to the secret decryption key when the user logs in. Operating an own server usually requires advanced technical knowledge and a dedicated device that is permanently online. Even if the latter could be a low-budget solution such as running the server software on a home-router or a small single-board computer, it is still an investment and for most users a significant hurdle to setup.

In the category of completely decentralized systems, which is the focus of this thesis, Retroshare[9] is an example of a deployed DOSN. It builds on a friend-to-friend network structure where users establish SSL connections to all their online friends. A DHT holds the IP addresses of friends to maintain connectivity even when IP addresses of the friends change. It features file-sharing, direct messaging and group chats and routes connections only via trusted links between friends, using the Turtle protocol [106]. There is, however, no replication of user profiles or persistent wall objects, so for most interactions between two users, both have to be online. The strict security policy of only trusting friend-to-friend connections provides good defenses against denial of service attacks and spam.

Another implemented system is a decentralized Twitter clone called Twister [57][10]. Twister is a completely distributed P2P microblogging platform, allowing users to post tweets and follow other users or hashtags. It also features end-to-end encrypted direct messaging between users that follow each other. Twister has an innovative user registration that leverages the Bitcoin blockchain technology to bind usernames to cryptographic keys, as mentioned earlier. Furthermore, it uses a DHT that stores user profiles and serves as a tracker for follower swarms. These swarms are sets of online users that follow one specific other user and a BitTorrent[11]-like protocol is used to deliver instant notifications about new posts and direct messages to these users. The DHT and local user storage serves as backup for the posts of a user and guarantee their availability even for times when no other user is following them. The plugin Sone for Freenet[12] is another example for a deployed decentralized social networking platform.

[9] `https://retroshare.github.io/`, retrieved 2016-11-01
[10] The software is available at `http://twister.net.co/`, retrieved 2016-11-01
[11] `http://www.bittorrent.org/`, retrieved 2016-11-01
[12] `https://wiki.freenetproject.org/Sone`, retrieved 2016-11-01

Focus of this Thesis

One main focus in research on DOSNs has been on content confidentiality and removing the threat that the central provider poses. While this is an important part of developing more privacy-preserving alternatives to centralized OSNs, the problematic effects of distributing the power of the provider over several entities and the possibilities for inferences that persist despite content encryption have often been neglected. Our first contribution in Article A is therefore a systematic overview of the new adversary models and challenges for user privacy in this changed context. It stresses the importance of metadata, that can endanger users' privacy even if all content is encrypted. Having these new threats in mind, we proceed in Articles B, C and D with proposing DOSN protocols for common OSN functionalities such as password management, user search and event invitations. With these contributions, we do not aim to build complete or self-contained DOSNs, but focus on one functionality. We therefore make a range of assumptions for these articles, such as a distributed storage with some properties being available and an identity management system being in place. Article E is a more applied contribution and takes up a specific instance of metadata privacy issues in a deployed distributed system – website fingerprinting attacks in the Tor anonymization network – and discusses threats and countermeasures to this specific problem.

4. Article Overview and Contributions

4

This thesis is comprised of five articles, originally published in peer-reviewed conference and workshop proceedings. In this section I describe their research questions and methodology, summarize the content of the articles and give account of the contributions I have made to each of them. Finally, I list articles that I co-authored during my PhD studies that are not included in this thesis.

General Research Question and Methodology

How can Internet-based services, in particular those of Online Social Networks, be decentralized to improve user privacy? This is the overarching research question of this thesis. It is broken down in more concrete questions for the different articles that this thesis is comprised of.

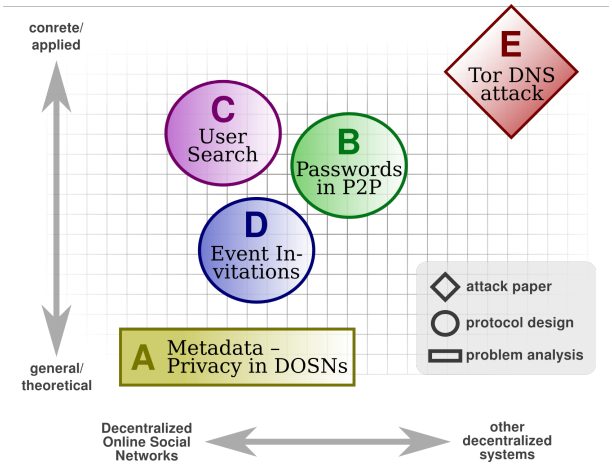


Figure 4.1: Categorization of the articles included in this thesis.

The included articles have varying degrees of generality and differ in how much they focus on DOSNs or other decentralized systems. Figure 4.1 summarizes these differences by roughly localizing the articles with respect to their relative positions on these dimensions. The first one, Article A, is a problem statement, systematically listing the new challenges to user privacy that arise from decentralization. Articles B, C and D, have a focus on protocol design, to show how different OSN functionality can be implemented in a decentralized and privacy-preserving way. The last one, Article E, analyzes a specific attack on a deployed system, the Tor anonymization network, shedding light on a very concrete problem of a decentralized system and possible solutions.

In the field of Computer Science, having roots in mathematics as well as engineering, different cultures of research methodologies can be found. On the one hand the ones inspired by mathematics with logical argumentation, formal models and proofs, and on the other hand engineering approaches with case studies, experiments and measurements [43]. The research methodology used for the work in this thesis can be located more in the latter culture and is probably best described by the design science research paradigm. While in social and natural science the traditional research paradigm is a descriptive one of creating testable theories, the design science research methodology aims at creating artifacts that are explicitly applicable to a problem in research or practice. It can be described as a process with six steps: problem identification and motivation, defining objectives of a solution, design and development, demonstration, evaluation, and communication [104]. The first article contained in this thesis, Article A, has a focus on the first two steps, problem identification, motivation and defining the objectives of a solution. The protocol designs in Articles B-D have their focus on the next three steps, design and development, demonstration and evaluation. The last article, Article E, is more self-contained and can either be seen to be a partial evaluation of the Tor anonymization network or to be a process on its own of designing and evaluating an attack on the system. The last step in the process, communication, was achieved by publishing the conducted work.

Article A: The Devil is in the Metadata – New Privacy Challenges in Decentralised Online Social Networks

originally published as *B. Greschbach, G. Kreitz, and S. Buchegger, "The devil is in the metadata – New privacy challenges in Decentralised Online Social Networks," in IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), March 2012, pp. 333–339.* For this thesis, Figure A.2 was enhanced both graphically and content-wise.

When moving from a centralized to a decentralized OSN design, the problem of one central data aggregation point vanishes. At the same time, new challenges to secure user privacy arise, that had not been there in centralized systems or were

not part of the threat model, because the central provider was assumed to be a neutral and trusted party in these systems.

In this article, we map out these new challenges in a systematic way. We look at possible privacy breaches stemming not from the content itself but from its metadata (like size or structure) or data handling (such as communication flows). Furthermore, we discuss the role of different adversaries in DOSNs. Finally, we list possible countermeasures and general approaches to mitigate these problems.

Specific Research Question and Methodology

This article tries to answer which negative effects decentralization can have on user privacy and how these effects can be mitigated. For mapping out the problem space, episodic evidence of privacy problems stemming from decentralization were gathered from related work (e. g., that already the existence of data can give away sensitive information), and used to seed a general hierarchy of possible problems (e. g., creating the main category "storage" with a subcategory "existence"). This hierarchy was then complemented by our own analysis to map out the possible problems as comprehensively as possible. For mapping out the different adversary models a similar approach was taken. As no experiments were conducted and the focus was on mapping out a problem space, this work can be seen as a theoretical contribution to formulate a problem statement for further research.

My Contributions

The main idea for Article A evolved from joint discussions with Sonja Buchegger. The classification of threats and adversaries was mainly developed by me with helpful input from Sonja Buchegger and Gunnar Kreitz. I did the bulk part of writing with substantial input from the co-authors.

Article B: Passwords in Peer-to-Peer

originally published as *G. Kreitz, O. Bodriagov, B. Greschbach, G. Rodríguez-Cano, and S. Buchegger, "Passwords in peer-to-peer," in IEEE 12th International Conference on Peer-to-Peer Computing (P2P), September 2012, pp. 167–178.*

When decentralizing an OSN service, two challenges arise. First, to implement the OSN functionality in a decentralized way without making use of centralized, trusted third parties. Second, to guarantee user privacy under the changed threat model, as outlined in Article A.

This article illustrates an instance of this two-folded challenge. We look at the password login and management feature of common OSNs and discuss how this can be realized in a DOSN. So we both have to come up with a decentralized implementation of the different functionalities and take care to make it secure and privacy-preserving under the threat model present in a P2P network.

More specifically, we propose and evaluate distributed protocols for password login, account registration, password change, remembered logins, logout of a remote device, and password recovery using security questions or e-mail notifications.

Specific Research Question and Methodology

The research question behind this article is, how a classical username–password login mechanisms can be transferred to a decentralized P2P system. Concerning the research method, this is a protocol design and analysis article. We first assembled a list of requirements, informed by standards for password-based authentication, related work and additional requirements that we considered relevant for the decentralized setting. Next we designed protocols that implement different functionalities (e. g., password change or password recovery) and that fulfill the compiled requirements. We formalized these protocols in pseudo-code. Then we ran experiments that simulated these protocols on DHT latency data to get performance benchmarks and did calculations for analyzing the security guarantees of the suggested protocols. We did not formally proof the security properties of the protocols, mainly due to lack of time, but also because a comprehensive formal verification makes more sense when deploying the protocols in a specific system where the concrete properties of the underlying components are known and fixed, for example which security guarantees the distributed storage provides.

My Contributions

The main work for this article was done by the first author Gunnar Kreitz. I contributed with ideas in joint discussions and the pseudocode formalizations of the protocols, proposing several improvements.

Article C: User Search with Knowledge Thresholds in Decentralized Online Social Networks

originally published as *B. Greschbach, G. Kreitz, and S. Buchegger, "User Search with Knowledge Thresholds in Decentralized Online Social Networks," in Privacy and Identity Management for Emerging Services and Technologies, M. Hansen, J.-H. Hoepman, R. Leenes, and D. Whitehouse, Eds. Springer Berlin Heidelberg, 2013, pp. 188–202.*

In this article, we look at user search, another standard feature of OSNs. There is a trade-off inherent in this feature, where user privacy has to be balanced with the findability of a user. We use a user-defined knowledge threshold to allow users to adjust this balance ("find me if you know enough about me") and propose distributed protocols to implement this feature in a DOSN.

We evaluate our protocols using real world data to relate the performance for legitimate users who try to find another user, to the costs of an adversary who attempts to acquire unknown information not intended for her.

Specific Research Question and Methodology

The lead question of this article is, how user search can be realized in a DOSN where, in the absence of a central trusted party, the findability and privacy requirements of users need to be balanced. This is another protocol design paper, where we first defined the functionality that we want to implement in a DOSN (user search with knowledge threshold) and then design protocols that implement this feature. We first worked out one possible implementation of the protocol. Informed by an analysis of this implementation we designed an alternative implementation that solves some shortcomings of the first version. We formalized both protocol implementations in pseudocode. As a result of an informal security analysis we identified one likely attack. To show how the protocols protect users against it, we conducted a brute-force simulation using real-world demographic data.

My Contributions

For this article, I was the first author and developed the ideas out of joint discussions with the co-authors. Together with Gunnar Kreitz I developed the protocols and evaluations, with important input from Sonja Buchegger. The people mentioned in the acknowledgements contributed with helpful ideas.

Article D: Event Invitations in Privacy-Preserving Decentralized Online Social Networks

originally published as *G. Rodríguez-Cano, B. Greschbach, and S. Buchegger, "Event Invitations in Decentralized Online Social Networks" in Privacy and Identity Management for the Future Internet in the Age of Globalisation 2014, Jan Camenisch, Simone Fischer-Hübner, and Marit Hansen, Eds. Springer Berlin Heidelberg, 2015, pp. 185–200.* (received the Best Student Paper Award)

Another functionality that is more specific to OSNs is event invitations. That is a feature that allows an organizer to invite a set of users and get feedback on who of them committed to attend. In this article, we formalize desired properties for an event invitation feature, such as who can see the identities, or only the number of invited or attending users. We then develop a set of privacy-enhancing tools based on cryptographic standard techniques that allow us to implement this feature in a completely decentralized way. Storage location indirection is, for example, used to control who can access the ciphertext of an encrypted object by storing only an encrypted link at the object's storage address. The encrypted object itself is then stored at the obfuscated storage location pointed by the link. This discloses

metadata about the encrypted content, such as size and modification history, only to selected users. We further develop a commit-disclose protocol, that allows an organizer to disclose some information only to users who committed to attend the event. Using these protocols and tools we show how an event invitation feature can be implemented with a wide range of possible privacy settings and in a decentralized environment, that is without any central party and with minimal trust assumptions in the involved parties.

Specific Research Question and Methodology

The research question behind this article is, how an event invitation feature can be implemented in a DOSN, allowing for different privacy settings. This paper focuses again on protocol design. We first formalize a set of security and privacy properties for this more specific feature of event invitations. We then introduce a general implementation framework and go exhaustively through all possible combinations of privacy settings that can be chosen according to the defined privacy properties to show how they can be implemented in this framework. We discuss a set of general techniques that we designed for the protocol implementation and that might be useful in other P2P applications as well. Finally we discuss the security of the protocols.

My Contributions

Together with Guillermo Rodríguez-Cano, I developed the problem statement, the formalizations, the cryptographic tools, protocols and analysis for this article. Sonja Buchegger contributed with helpful feedback and writing.

Article E: The Effect of DNS on Tor's Anonymity

accepted for publication as *B. Greschbach*, T. Pulls*, L.M. Roberts*, P. Winter*, and N. Feamster, "The Effect of DNS on Tor's Anonymity" in Network and Distributed System Security (NDSS) Symposium 2017.*

*All four authors contributed substantially, and share first authorship. The names are ordered alphabetically.

The Onion Router (Tor) is a decentralized, low-latency anonymization mix-network that allows a user for example to browse to a website without anyone learning both the user's IP address and which website the user browsed to. A user's Tor client first establishes an encrypted connection to an exit relay of the network, but redirects this connection through two other relays, an entry guard and a middle relay, so that the exit relay does not learn the user's IP address. The exit relay is then instructed to establish a connection with the destination website, which only learns the IP address of the exit relay. The Tor network is constantly improved to protect against new attacks. One of these attacks is website fingerprinting, where

a passive adversary analyzes the ciphertext of the user's connection with the entry guard to guess the requested website using machine learning techniques based on features such as package inter-arrival times and package payload of the encrypted connection.

In this article, we explore a new kind of website fingerprinting attack incorporating DNS data that adversaries might be able to observe from exit relays. We first analyze how exposed Tor DNS traffic is: we determine which DNS resolvers are used by exit relays and which autonomous systems (ASs) the DNS traffic traverses. We found that for many websites DNS traffic traverses more than two times more ASs then the corresponding TCP traffic and some organizations that operate public DNS resolvers, such as Google, see almost 40% of all Tor DNS traffic. We further analyze what impact these findings can have on the Tor users' anonymity by developing a set of correlation attacks that use observed DNS data. We find that these attacks can increase an attacker's precision significantly and discuss possible counter-measures to mitigate this threat.

Specific Research Question and Methodology

The research question of this article was, how the handling of DNS requests in the Tor anonymization network affects the anonymity of its users. Regarding research methodology, this article is rather different from the other included articles as it focuses on a specific attack on a concrete, deployed system. We start with analyzing how DNS requests are handled in Tor (what resolvers are used by exit operators) and what effect this has on possible adversaries (quantifying the exposure of DNS requests to passive network adversaries by analysing the network routes the requests take). We then explore how DNS traffic could be used by an attacker to improve website fingerprinting attacks. To measure the effects of these combined attacks, we simulate the Tor users' website access patterns based on Tor statistics, our own measurements on an exit relay under our control and figures from related work. We evaluate the effects of the combined attack on precision and recall, investigating the impact of different parameters such as Tor network size, weight-learning round for the machine-learning algorithm or the popularity of the visited website. Based on these results we finally discuss strategies how to mitigate the effect of this attack.

My Contributions

For this article, I mainly worked on the attack part (Section V) together with Tobias Pulls. My main contributions were in joint discussions on the design of the correlation attacks, and the modelling of the Tor network's web traffic (Section V.A and V.B).

Articles not Included in this Thesis

During my PhD studies I additionally co-authored or authored the following published articles that are not included in this thesis:

- *Location Privacy in Relation to Trusted Peers.* Klaus Rechert, Benjamin Greschbach. 7th International Workshop on Security and Trust Management 2011 (STM'11).
- *Assessing Location Privacy in Mobile Communication Networks.* Klaus Rechert, Konrad Meier, Benjamin Greschbach, Dennis Wehrle, and Dirk von Suchodoletz. 14th International Conference on Information Security 2011 (ISC'11).
- *Location Privacy: User-centric Threat Analyis* (student session abstract). Benjamin Greschbach. 7th European Conference on Computer Network Defense 2011 (EC2ND'11).
- *Friendly Surveillance - A New Adversary Model for Privacy in Decentralized Online Social Networks.* Benjamin Greschbach and Sonja Buchegger. 5th interdisciplinary Conference on Current Issues in IT Security 2012.
- *Exploring Decentralization Dimensions of Social Networking Services: Adversaries and Availability.* Thomas Paul, Benjamin Greschbach, Sonja Buchegger, and Thorsten Strufe. First ACM International Workshop on Hot Topics on Interdisciplinary Social Networks Research 2012 (HotSocial'12).
- *Design of a Privacy-Preserving Document Submission and Grading System* (short paper). Benjamin Greschbach, Guillermo Rodríguez Cano, Tomas Ericsson, and Sonja Buchegger. 20th Nordic Conference on Secure IT Systems (NordSec'15).

Finally, this doctoral thesis is based on and extending my licenciate thesis that already included the first three articles, Article A, Article B, and Article C:

- *Privacy Analysis and Protocols for Decentralized Online Social Networks.* Benjamin Greschbach. Licentiate Thesis, KTH Royal Institute of Technology, 2015. ISBN 978-91-7595-546-9.

Published Articles

Article A

A

The Devil is in the Metadata – New Privacy Challenges in Decentralised Online Social Networks

Benjamin Greschbach, Gunnar Kreitz, and Sonja Buchegger

KTH Royal Institute of Technology
School of Computer Science and Communication
Stockholm, Sweden
{bgre, gkreitz, buc}@csc.kth.se

Abstract

Decentralised Online Social Networks (DOSN) are evolving as a promising approach to mitigate design-inherent privacy flaws of logically centralised services such as Facebook, Google+ or Twitter. A common approach to build a DOSN is to use a peer-to-peer architecture. While the absence of a single point of data aggregation strikes the most powerful attacker from the list of adversaries, the decentralisation also removes some privacy protection afforded by the central party's intermediation of all communication. As content storage, access right management, retrieval and other administrative tasks of the service become the obligation of the users, it is non-trivial to hide the metadata of objects and information flows, even when the content itself is encrypted. Such metadata is, deliberately or as a side effect, hidden by the provider in a centralised system.

In this work, we aim to identify the dangers arising or made more severe from decentralisation, and show how inferences from metadata might invade users' privacy. Furthermore, we discuss general techniques to mitigate or solve the identified issues.

Introduction

As people use Social Network Services (SNS) to organise their social life, privacy issues are an inherent concern in these services. Currently, a user must trust the SNS provider to enforce access rights management, not to misuse the provided content, and to be sufficiently secured against third-party attacks. For today's popular SNS providers, however, "people are not customers, but primarily products" [65]. Their business model is based on targeted advertisements, and they have an infamous history of data leakages and privacy breaches.

In response to these shortcomings, Decentralised Online Social Networks (DOSN) have been proposed. There is a wide range of designs spanning from centralised to decentralised network architectures. In the completely decentralised approaches, the users themselves form a peer-to-peer (P2P) network in order to collaboratively provide the storage and communication infrastructure for the social network service. Access control for published content is enforced by cryptographic means so that users need not rely on policies or the benignity of a central provider. In addition, users keep the physical ownership of their content, which prevents censorship, yields higher resilience with respect to network outages, and facilitates data portability.

When solving the privacy issues of the centralised system by moving to a decentralised design, however, new privacy challenges arise. Simply encrypting the content is not enough to hide all sensitive information from attackers, and although a powerful central provider is not present in these kinds of systems, several other adversary models become relevant.

We remark that although several of the issues raised in this paper have been previously mentioned in the literature, the focus in SNS privacy research has been on content confidentiality and on removing the threat that central SNS providers pose. While this was an important development, we believe that the logical next step is to systematically study the effect of distributing the power of the provider over several entities and examining the possibilities for inferences that persist despite content encryption, including traffic analysis issues in this new context. Failing to protect against even a single one of these threats can lead to serious privacy breaches in an otherwise secure system.

Our Contributions

In this paper we highlight the new privacy challenges that arise once a centralised SNS is replaced by a DOSN. Specifically, we systematically discuss possible privacy breaches stemming not from the content itself but from its metadata (like size or structure) or data handling (such as communication flows). Furthermore, we discuss the role of different adversaries in DOSNs. Finally, we summarise approaches to mitigate these problems, including those suggested by proposed DOSN implementations. To the best of our knowledge there is no solution dealing with the whole range of the problems we discuss.

Paper Outline

The rest of the paper is organised as follows. After referring to related work in Section A.2, we sketch the different models of SNS implementations including relevant attackers in Section A.3. Next, Section A.4 lists the possible metadata privacy leakages, that is sensitive information which can be inferred even when the content does not leak. Section A.5 discusses countermeasures to approach these new challenges before Section A.6 concludes the paper.

Related Work

Research related to the scope of this paper can be found in mainly three areas: privacy issues in SNS, decentralised online social networks, and metadata privacy in general.

The impact of SNS on their users' privacy has been extensively studied. Gross et al. [64] have identified several threats of SNS usage such as stalking; de-anonymisation of external sensitive sources such as anonymised medical records; identity theft, e. g. by social insurance number reconstruction; user profiling by building a digital dossier and simplified social engineering. Danezis et al. [40] point out that the position of a user in a social network reveals characteristics about the person, such as their status and potential influence reach. Paul et al. [102] underline the consequences of massive central data aggregation in conjunction with an advertising-based business model of major SNS providers. They warn against the risks of direct misuse or unintended leakage of this data that is not appropriately protected and hard to anonymise. Krishnamurthy and Wills [84] show that relevant leaks do occur in practice.

One main approach to address the privacy issues in SNS is decentralisation. Buchegger et al. [27] propose the *PeerSoN* system where (encrypted) content is distributed using a P2P network formed by the users of the SNS. Aiello and Ruffo [6] elaborate on a Distributed Hash Table (DHT) based architectural framework supporting SNS functionality. They propose authentication on the routing level and discuss implementations of SNS requirements such as access control, reputation management and search operations. Cutillo et al. [39] introduce *Safebook*, an architectural approach focusing on communication anonymisation. Content is stored at trusted friend nodes and requests are routed through a mix-network formed by social links to obfuscate information flow. Sharma and Datta [120] describe *SuperNova*, that is based on a hybrid network architecture where highly available "super peers" are used for crucial tasks such as helping new users to join the network. The *Persona* project by Baden et al. [14] proposes the use of an attribute-based encryption scheme for social network operations. Finally, Bodriagov and Buchegger [22] scrutinise the proposals for DOSN-tailored encryption schemes and evaluate their performance for SNS operations.

One instance of a metadata privacy leakage in the context of SNS is mentioned by Anderson et al. [10]. They point out the threat of a friend learning about the

existence of content she does not have access to. Chew et al. [34] identify three possible leakages in SNS that are not caused directly by the disclosure of content: entries in the user's activity stream that were automatically generated based on the user's activities (also on third party sites); unwanted linkage of different sets of user data; and identifying inferences by merging social graphs. Traffic analysis, extracting and inferring information from network metadata, (see e. g. Danezis and Clayton [40] for an overview) is one of the attack techniques we consider.

Decentralising Social Networks

On an abstract level and following a minimalistic definition (e. g. [85]), we assume an SNS to be merely an integration of user generated content with social relationship information.

The latter is used mainly for access control, data presentation, and friendship announcements. Content comprises all active contributions of a user to the system, static (such as profile attributes) as well as more dynamic ones (such as status updates, text-, picture-, video- or link-posts). It also includes interactions such as comments or simple like-indications in response to posts, enrichments of posts with social links (e. g. *tags* in pictures) as well as asynchronous or synchronous messaging (e. g. private messaging, chats). Timestamped notifications about this data are usually automatically pushed to the user in a "news feed".

In a concrete implementation of such a system, the degree of centralisation of control over user data is an architectural design choice that impacts both possible privacy leaks and types of attackers.

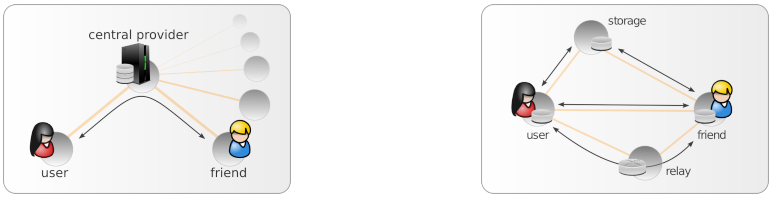
Architectures

Considering the proposed DOSN implementations in the literature, one can observe a broad range of topologies rather than a bipartite division between fully centralised and fully decentralised systems. Several hybrid approaches (e. g. *Diaspora*[1]) use dedicated, semi-trusted nodes to address availability, bootstrapping and other issues that are difficult to solve in a flat P2P network. For the rest of this paper, however, we focus on the differences between the two extreme cases of logically centralised designs (*Facebook* or *Google+*) and completely decentralised approaches (*PeerSoN*, *Safebook*, or *Persona*).

In the centralised case, the relevant agents are the SNS provider and the users of the SNS, with all communication between users relayed by the central provider. In the decentralised approach, the provider is replaced by a P2P network formed by the users.

P2P approaches are varied, but here we sketch a simplified example system. A user herself hosts all content she posts. To ensure availability even when she is offline, her content is also replicated by a number of storage nodes. Access

[1] `http://diasporaproject.org/`



(a) All communication is relayed by the central provider.

(b) Besides direct peer communication, several nodes can be involved.

Figure A.1: Communication flows in a) centralised and b) decentralised SNS

control can be implemented either by having the user and storage nodes requiring authorisation to serve data, or by encrypting objects such that only authorised users can decrypt the content. We focus on the latter type of design, where any user can request any encrypted data. This means that storage nodes need not be trusted and need not be informed of ACL rules for the content they replicate.

Another type of service provided by nodes to each other in the P2P scenario is relaying traffic in the overlay. This may include forwarding traffic for two nodes who cannot directly connect due to firewall restrictions, implementing a DHT, or to anonymise communication. We refer to nodes acting in such capacities as relay nodes. We illustrate the different abstract communication flows in Figure A.1.

Privacy Advantages of Decentralisation

The most important privacy advantage of a decentralised system is the absence of a central point of data aggregation. In the case of a centralised system with unencrypted storage, the provider can mine the data without limitations and infer information from both the content and metadata. In addition to deliberate privacy infringements, also by disclosure to third parties, centralised data collections are vulnerable to accidental leaks caused by inadvertent insider behaviour or attacks on the system. In a centralised system that employs user-side encryption to protect the content (e. g. *Pidder*[2]), the provider only observes metadata. When drawing inferences from it, the provider is, however, in the best position possible as it has a complete view of all users of the system at all times.

In a P2P system, data might be replicated by friends (e. g. *Safebook*) or random strangers (e. g. *SuperNova*), but no systematic accumulation of user data occurs.

New Challenges from Decentralisation

While removing the single point of data aggregation constitutes a general advantage of more decentralised architectures, there are also several drawbacks and new

[2] `http://pidder.com/`

privacy challenges when building on a P2P network. In a centralised system the users' content is entrusted to a single party that only gives access to entitled principals. Deliberately or as a side effect, this intermediation procedure hides metadata information from requesting users. Thus, while the operator of a centralised system can learn significant information from metadata (and content, if not encrypted), such information is hidden from everyone else.

In the decentralised systems we consider, several parties are involved in storing and communicating user content, and authorisation is performed via encryption of the data. This aggravates the problem of metadata privacy leakage because more parties can access such information as illustrated in several examples in Section A.4. Unless the decentralised system is carefully designed, it may admit similar privacy invasions from peers in the system or third parties requesting large amounts of data as were possible by the central provider, thus weakening the privacy motivation for selecting a decentralised design.

A new threat that arises from metadata in a decentralised system is that of a more powerful friend adversary (an attacker that exploits its social ties to the user). One feature of the friend adversary contributes eminently to this problem: friends have more background knowledge related to the user – not necessarily acquired only via SNS communication – that enables them to accomplish effective inference attacks even on sparse raw data. If a friend for example knows about a couple of preferred places the user usually visits, coarse IP address based location information suffices to determine the user's exact geographic location with high probability.

Additionally, traffic analysis yields more information in a decentralised system where information is exchanged directly between communicating parties. The intermediation of very high volumes of communication via a few data centres by centralised solutions serves to hide traffic patterns against outside adversaries (but not against the provider).

In a fully decentralised setting, it is also more difficult to enforce a limit on the rate at which data can be requested. This may allow multiple third parties to collect significant amounts of public information from the DOSN. While such information is by definition public, aggregating and indexing a massive amount of it can constitute a privacy invasion.

Adversary Models

We distinguish between different adversaries in the context of SNS by their functional power resulting from their role and position in the network.

Relay nodes and **storage nodes** can make use of their special role and position in the network. Relay nodes can easily observe all traffic they forward for other nodes, and storage nodes can analyse the data entrusted to them as well as log all requests they receive. **Friends** of a user – or other socially close nodes like friends of a friend – can try to obtain more information than what the user chose to share with them. This can be done by exploiting the way data storage, encryption and communication is implemented in a DOSN. Having additional background knowl-

edge about the user and possibly incentives for targeted attacks can turn a friend into a powerful attacker. Network **sniffers** who observe communication traffic at an arbitrary location in the network constitute another category of possible attackers. **Harvesters** are nodes that simply request data from the system to learn from the metadata they receive.

In order to compare the decentralised system architecture with the centralised SNS, we also list the **central SNS provider** as an adversary. If present, it constitutes the most powerful attacker possible because it observes all content and communication from all users of the system. Even if the content is encrypted, the provider still has a complete picture of communication traffic and content metadata.

The adversary types discussed here have access to different data. Here, we consider five categories of data that an adversary may exploit. An adversary may learn **access patterns**, that is information about when content is requested or modified. She may be able to access **ciphertext representations** of content. She may have intimate **background knowledge** about the victim. She may see all or a fraction of the victim's **network traffic**, and be able to relate it to the victim, which we refer to as **micro-scale** network access. Finally, she may have a global but incomplete view of network traffic in the system that we call **macro-scale**.

Table A.1: Adversary Capabilities

	Relay	**Stor.**	**Friend**	**Sniff.**	**Harv.**	**Cent.**
Access pattern		✓				✓
Ciphertexts		✓	✓		✓	✓
Backgr. know.			✓			
Net, micro	✓					✓
Net, macro				✓		✓

We summarise the capabilities of adversaries in Table A.1. From this overview, it can be seen that no single class of attacker is as powerful as the central adversary, but unless the system design adequately addresses metadata concerns, the new attackers may be as powerful as the central one. Moreover, it is significantly easier to position oneself as an adversary in a decentralised system than in a centralised one.

Collusions of several agents in the network also need to be considered. This includes a single agent having several of the roles outlined above (e. g. being both a friend and a storage node), and an adversary paying the cost to operate a large number of nodes.

Inferences from Metadata

By metadata privacy leakages we mean disclosures of sensitive personal information that do not stem from the content of published data but from properties of it (such

as size or structure) or information generated while managing it (like communication flows).

Possible inferences from metadata can invade a user's privacy in the same way as sensitive personal information obtained from posted content. This includes identifying information (directly or indirectly), general descriptive data (interests, political attitudes, health condition, etc.) as well as more SNS-specific information such as social relationship data (number of friends, nature of relations, etc.), or behavioural data (activity, location, etc.).

While the DOSN approach is a substantial improvement compared to common centralised systems, we want to illustrate which threats to the users' privacy still remain and which new challenges arise. In the following we assume the SNS to be decentralised with all content and communication encrypted. We further assume that it is correctly implemented and perfectly protects the content. Besides that, we only consider a naïve design of the DOSN and individual worst cases in order to give a comprehensive overview of the possible problems. That implies that there are easy fixes for some of the raised issues – this, however, is discussed later in Section A.5. We have chosen not to study any particular proposed system, as source code for these is not generally available, or only in beta version.

Inferences from Stored Content

While encrypting the content solves many important privacy concerns, there still remain possibilities of privacy leakages from the stored data. The size, structure, and (implicit) modification time of the ciphertexts may reveal information that the user originally intended to hide. In the following we give examples for each of these properties.

Size

The size of an object's ciphertext is an indicator for the content type of the stored object (e. g. like-flag, text, image, video). Additionally, characteristics such as an estimated word count or the length of a video can be inferred. Moreover, the size of larger files, such as video, may be reasonably unique (at least among objects posted during a given time period or from a specific region). Such uniqueness could allow the ISP of a regime sniffing the network to trace which users have re-shared a forbidden video on the DOSN by simply looking for posts of objects with the exact same size as the video at issue. This type of attack requires access to **ciphertexts** or **network** traffic, either **micro-scale** or **macro-scale**.

Structure

An adversary may infer not only the size of single objects, but also statistical information about a set of objects of a certain kind, like the number of objects in a list (e. g. unencrypted documents with data references in *Persona*). Linking

this knowledge with information about the content type leads to another form of metadata privacy leakage – revealing the number of pictures in an album or the number of comments to a post. This attack requires access to **ciphertexts**.

Assume for example a user sharing pictures of her recent holidays with friends. Being asked for them at work, she decides to grant access to a subset of them to a colleague. Inferring from the data structure that there are more objects in the album than he can decrypt, the colleague learns the exact number of pictures that are hidden from him.

Modification History

Once the storage location of a specific object and some general information about its type are identified by an adversary, monitoring the ciphertext for changes reveals possibly sensitive information. The modification history can for example tell something about the frequency of a user's status updates, the intensity of her commenting activity, or other general usage patterns. This attack requires **access pattern** information, or, with polling, access to **ciphertexts**.

Assume a friend observes frequent modifications of an object, identified as the user's encrypted status update representation. While the version displayed to her does not change, the friend learns that she is excluded from at least parts of the user's updates.

Inferences from Access Control Mechanisms

One reason for a user to provide social relationship information is the realisation of fine-grained access control mechanisms. Depending on the implementation of these mechanisms, the chosen access right settings might allow conclusions about a user's social relationships to be drawn, as we outline in this section.

Encryption Header

If an access control list (ACL) or other cryptographic key material is stored together with the encrypted object – e. g. in a prepended header – the size of this header can allow inferences about the identity or number of individuals who can access the content. This attack requires access to **ciphertexts**.

Exploiting the same feature either for a central object of a user – like her wall representation – or a representative set of content objects belonging to her, can reveal the total number of friends the user has.

Key Distribution

Adding a new friend or revoking access rights of an existing friend will – depending on the encryption scheme and implementation – trigger re-keying and/or key distribution mechanisms that can be observed even by users who are not subject

to the relationship change itself. This attack requires **micro-scale network** access. When combined with **background knowledge**, significantly more revealing conclusions can be drawn.

Assume a user expels another user from her circle of friends. If a new group key is sent to her remaining friends, an adversary observing this revocation can, together with background information about the user's social relationships, infer the specific person that was removed.

Key Reutilisation

If the same key or encryption header is used for several objects, even adversaries who cannot decrypt the content, trivially learn that the same access rights are in place for these objects. This information can be exploited in several ways. Mapping out relations for a large number of objects might allow inferences about the structure of a user's friend circle. A friend, who has access to the objects and observes another user reacting to one of them (e. g. by a comment or a like-flag), immediately learns that this user has access to all the other objects as well. This attack requires access to **ciphertexts**. **Background knowledge** enhances the attack.

Inferences from Communication Flows

An adversary can gain additional insights into a user's activities by capturing network traffic that is related to the user. This might be performed by an external network sniffer as well as persons related to the user, e. g. a node that is hosting some of the user's content and observes the access logs.

Direct Connections

SNS-related network traffic can already on a very low protocol level (e. g. IP header information) reveal sensitive information. In the case of direct communication with the user's device – a common scenario in P2P architectures – the IP address of this user is trivially obtained and can be tracked over time. This allows correlating with activities of the user on other internet services like file sharing or voice-over-IP (possible even when located behind a NAT, see [21]), determining geographic location information about the user via geo-IP mappings, or inferring general usage patterns, such as the user's online times or working habits (when does the user connect from which device). This attack makes use of **network** access, either **micro-scale** or **macro-scale**. **Background knowledge** allows more precise conclusions to be drawn.

Content Requests

The access logs of content that an adversary is hosting or providing to the user disclose the user's requests for specific objects – therefore acting as implicit reading receipts for new content – and might allow general profiling of the user's interests.

Moreover, observing a set of users' access patterns has the potential to identify the ownership as well as possible access rights of content objects. Companies may find it profitable to operate a large number of storage nodes in order to monitor requests. For instance, an insurance company may attempt to identify users accessing content posted in groups related to cancer or other diseases. This attack requires **access pattern** information, or **network** access, either **micro-scale** or **macro-scale**.

Content Sharing

Storage nodes as well as sniffers that capture traffic to these can easily observe upload activity. This includes the frequency of changes to stored content and might allow similar conclusions as sketched in Section A.4. Furthermore, timing-based inferences are a possible way to infer access rights if, for example, the distribution of key material to a set of other users is observed shortly after a new content object was uploaded. By monitoring the upload activity of several users, sniffers might moreover learn ownership relations between the stored content and the uploading users. This attack requires **access pattern** information, or **network** access, either **micro-scale** or **macro-scale**.

Control Messages

Depending on the protocol implementation, specific user operations such as login, adding friends, search requests, etc. can yield certain patterns of control messages a sniffer can observe and thus infer the kind of operation. The login procedure of a user may comprise polling friends for updates that happened while the user was offline, communicating with storage nodes or similar characteristic sequences of administrative operations. This attack requires **network** access, either **micro-scale** or **macro-scale**.

Countermeasures

There exist several approaches to mitigate the described metadata privacy leakages but to the best of our knowledge no comprehensive concept to cope with them all. In the following, we discuss solutions from the DOSN literature as well as from other fields.

Stored Content

To hinder inferences based on the size of ciphertexts, padding is one way to obfuscate the exact content length. That might help against fingerprinting objects by size but may still allow inferences about the content type from the order of magnitude. Another strategy could be to split up content objects into blocks of uniform sizes and hide their connection (e. g. [10]). The latter is, however, non-trivial especially against an adversary performing communication flow analysis.

To hide the structure of composite or related storage objects, an encryption scheme that conceals not only the content of the single objects but also indices and links is one solution. To not solely rely on encryption for authorisation, but use it as one of several layers is another approach. If semi-trusted storage nodes perform additional access right validations before delivering encrypted objects, adversaries not involved in storage cannot retrieve the ciphertext or the metadata information. However, this comes with the trade-off that the storage nodes must be given more explicit access-right information about the objects they keep. Additionally, dummy list entries and placeholder values for fixed fields can prevent an adversary from determining if values have been set in a user profile.

Assuming an insider adversary model (e. g. the storage node itself), hiding the modification history of a content object is very difficult. Baden et al. [14] propose to obfuscate the role of a storage object (e. g. a status update document) for that reason. Another way to conceal user-triggered changes can be the introduction of noise in the modification process, but dummy-change operations can be quite costly in terms of performance for an SNS system.

Access Control Mechanisms

Most of the presented access control related leakages can be approached with more sophisticated cryptographic schemes. In *Persona* attribute-based encryption (ABE) is used to realise group encryption without encrypting the symmetric content key with the public keys of all recipients. Groups defined by one user can even be reused by other users without them learning the explicit recipient list (and thereby enabling friend-of-a-friend access schemes). The attribute access structure stored with the object, however, might still allow inferences about the audience, e. g. by the attribute names carrying semantic meaning. The *PeerSoN* project suggest to use broadcast encryption schemes that have hidden access structures. Encryption headers in that case do not reveal anything about the audience of the content.

Communication Flows

In the literature, several protection mechanisms against communication flow analysis can be found – general ones as well as some explicitly related to SNS. Mix-network like communication anonymisation is a central part of the approach of the *Safebook* project. Information flows are obfuscated by routing them through a mix of socially related nodes, starting by those that are assumed to be most trusted. Caching can also mitigate communication flow leakages by minimising message exchange in general and decorrelating it from specific user actions. Obfuscation by noise – e. g. introducing dummy traffic – comes with the cost of higher network load but might be required in situations where other means are not applicable or not effective. Careful protocol design can help mitigate leakages as well by making control messages indistinguishable from content bearing messages. Another approach

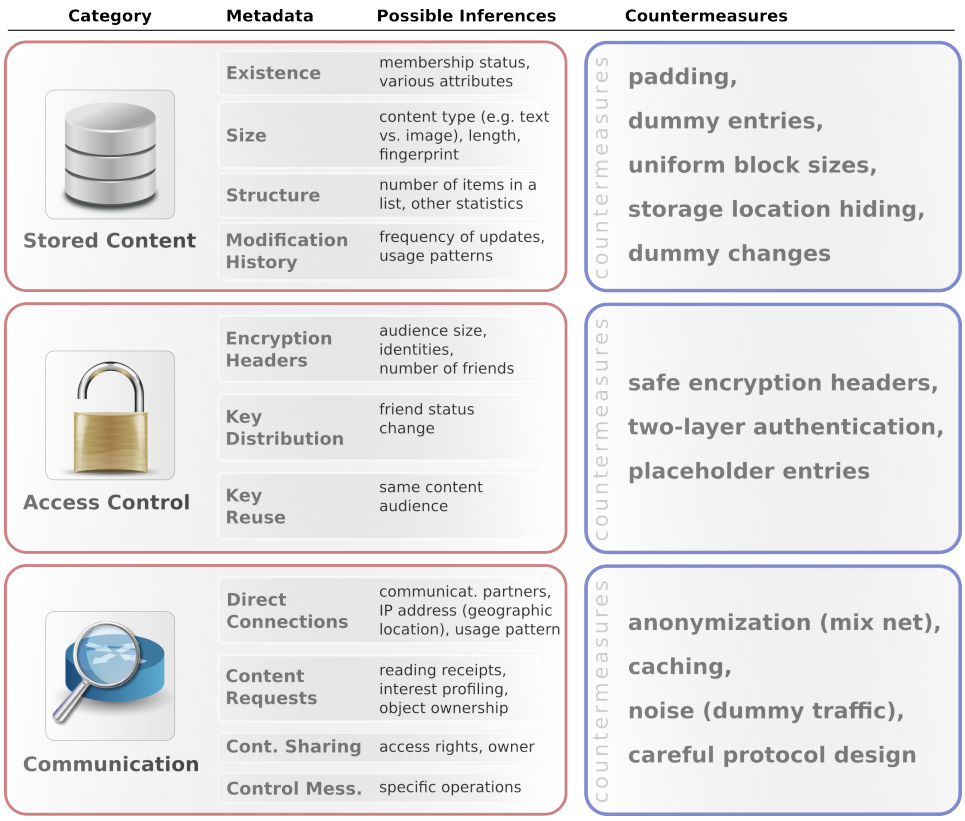


Figure A.2: Summary of metadata leakages and possible countermeasures.

is to make the DOSN protocol and communication patterns difficult to distinguish from some existing high-volume P2P protocol, such as *BitTorrent*.

Conclusion

Figure A.2 summarises the critical metadata in DOSNs and possible privacy leakages identified in Section A.4 as well as the approaches to tackle these problems discussed in Section A.5.

We conclude that while DOSNs have great potential to mitigate inherent privacy flaws of today's centralised SNS, simply encrypting the content is not sufficient. Metadata information like inferences from storage objects, access control mechanisms or traffic has the potential to expose the user to severe privacy threats. Furthermore, new adversaries enter the stage when the SNS has no single provider, as the decentralised network architecture exhibits more diverse points of attack. Some of these attacks are easy to protect against but when implementing a DOSN, these

issues have to be considered in a systematic manner in order to offer comprehensive privacy protection.

For future work, we plan to further investigate the special characteristics of the friend-adversary model. The aim is to gain a better insight into which inferences are possible for a socially close attacker in a DOSN setting where only sparse sensitive data but extensive background knowledge is available. Furthermore, we plan to evaluate the efficacy of the discussed countermeasures for concrete DOSN implementations when more mature code becomes available.

Acknowledgements

This research has been funded by the Swedish Foundation for Strategic Research grant SSF FFL09-0086 and the Swedish Research Council grant VR 2009-3793.

Article B

B

Passwords in Peer-to-Peer

Gunnar Kreitz, Oleksandr Bodriagov, Benjamin Greschbach,
Guillermo Rodríguez-Cano, and Sonja Buchegger

KTH Royal Institute of Technology
School of Computer Science and Communication
Stockholm, Sweden
{gkreitz, obo, bgre, gurc, buc}@csc.kth.se

Abstract

One of the differences between typical peer-to-peer (P2P) and client-server systems is the existence of user accounts. While many P2P applications, like public file sharing, are anonymous, more complex services such as decentralized online social networks require user authentication. In these, the common approach to P2P authentication builds on the possession of cryptographic keys. A drawback with that approach is usability when users access the system from multiple devices, an increasingly common scenario.

In this work, we present a scheme to support logins based on users knowing a username-password pair. We use passwords, as they are the most common authentication mechanism in services on the Internet today, ensuring strong user familiarity. In addition to password logins, we also present supporting protocols to provide functionality related to password logins, such as resetting a forgotten password via e-mail or security questions. Together, these allow P2P systems to emulate centralized password logins. The results of our performance evaluation indicate that incurred delays are well within acceptable bounds.

Introduction

Most of the peer-to-peer (P2P) systems deployed today do not authenticate users. While this is is often acceptable, or even preferable, there are some problems for which user authentication is a requirement. These include P2P storage, backup, and online social networks. In such applications, the data accessible to a client depends on who is using it.

We discuss how to implement a username-password scheme for authentication in P2P systems. Our goal is to construct an authentication component that can be reused across different P2P applications, which we assume authenticate via possession of cryptographic keys. Thus, from an API perspective, the login system shall allow a user entering a username and a password to recover a set of cryptographic keys which can then be used by the actual application. These keys can also be updated as needed by the application.

The goal from an end-user point of view is to emulate current behavior of centralized password-based login mechanisms. More specifically, we include schemes to remember logins, change passwords, and provide recovery if a password is forgotten. We aim to follow best practice in password authentication, acknowledging that users often re-use passwords between systems. By remembered logins, we mean that a user can opt to have a device store information such that it can log in again without storing the user's password in plain text on the device. Similarly, password change requires knowing the old password, and for password recovery, the user is able to set a new password but does not learn her previous one.

Why password authentication?

There is a rich literature on various approaches to authentication, ranging from the traditional username-password pair to hardware tokens and biometry. Of these, the traditional view is that passwords should be replaced by some better mechanism. However, as argued by Herley and van Oorschot [68], despite significant research efforts into dislodging passwords, they are still by far the most common authentication mechanism today. Reasons for their prevalence include simplicity, price, and very strong user familiarity.

When authentication is required in the P2P setting, it is typically done via the security-wise stronger mechanism of generating and storing cryptographic keys on a user's machine. This approach is taken in systems such as OneSwarm [71], Safebook [39], and Tribler [2]. This works well until the user wants to access the service from a second device. To do so, she would need to transfer the keys, or assume a new identity. This is an added complexity and user-perceived drawback for P2P services competing against client-server systems.

One concern is that using passwords may lead to added security risks for skilled and security-conscious users who can easily copy keys between devices. However, nothing prevents such users from choosing passwords of similar strength as cryptographic keys. Another issue pertains to remembered logins, where one must

consider theft. We cannot prevent a thief from accessing the user's account, but with our protocols, the thief cannot change the user's password, and the legitimate user can always revoke the remembered credentials that are on the stolen device.

Our Contribution

We develop and describe a suite of protocols for password authentication in P2P networks: account registration, login, password change, remembered logins, logout also of a remote device, and password recovery, following best practices and adapting standardized criteria from centralized systems to P2P environments, and start a discussion on usable authentication in P2P systems.

Our password authentication is based on standard cryptographic techniques and can be used with standard P2P components. As a first step toward a security analysis, we discuss the security implications of our protocols. Then, we evaluate the performance of our protocols under various scenarios.

Paper Outline

We discuss related work in Section B.2, give a system overview in Section B.3 and outline our basic scheme for password-based login in Section B.4. We then describe password recovery mechanisms as extensions to the basic login mechanisms in Section B.5. Next, we discuss security in Section B.6 and report our evaluation results in Section B.7 before concluding in Section B.8.

Related Work

The subject of securely establishing stable identities in P2P systems has been previously studied, for instance by Aberer, Datta and Hauswirth [3]. The need for identities mainly arose from technical concerns, such as handling dynamic IP address assignment, or avoiding Sybil attacks [46]. Authentication of a node is done via a signature key, automatically generated and stored on the node.

As P2P systems began providing more complex functionality [71, 39, 2, 88], the need to authenticate *users*, rather than nodes, arose. It seems that often, authentication via a signature key has been carried over to this problem. While a solution of automatic identification of a node is preferable as long as users use a single device, equating a node with a user fails as users increasingly access services from multiple devices.

Illustrative is the case of backup systems, where an important use case is to restore data on a different system from where it was backed up. Here, two different approaches to authentication have been taken. All approaches build on encrypting backed up content, and the approaches vary in whether the keys are randomly derived [88], or derived from a password [38]. In the former case, a user must manually back the keys up, as these keys are required to restore the backup. The systems deriving a key from a password are related to our proposed protocol, and

use some related techniques. However, to the best of our knowledge, they do not consider the additional protocols required surrounding password authentication, such as remembered logins, and recovering lost passwords.

Some P2P storage systems also use techniques which are related to ours. For example, the DHT-based systems GNUnet and Freenet use keyword strings to derive a public-private key pair whose private key is used to sign data and the hash of the public key to identify the data in the storage. Both of these systems use a keyword string as a seed to a pseudo-random number generator that produces the key pair [36, 19]. Knowing only the memorable keyword string the user can store and retrieve information.

Related to forgotten passwords, recovery of information in a P2P scenario has been studied by Vu et al. [132] who proposed a combination of threshold-based secret sharing with delegate selection and encrypting shares with passwords.

Frykholm and Juels [58] proposed a password-recovery mechanism based on security questions very similar to our protocol for the same task. They offer better, information-theoretic security properties, something not applicable to our scenario. We treat the subject of password change, which is not applicable to their scenario, although their proposal could be extended to support password change using our techniques.

System Overview and Assumptions

We have designed our system around standard primitives, as depicted in Figure B.1. In particular, our protocols build on: a DHT [139, 74], for user lookup; a peer sampling protocol [73, 23] for randomly choosing peers; and a distributed storage [20, 110] for storing data required for our solution. Both the DHT and distributed storage are P2P protocols, run by the peers participating in the system. The storage could be implemented as a DHT, or even be the same as the user lookup DHT. However, we put different requirements on the user lookup DHT and the distributed storage, as detailed below.

To make the system flexible across different implementations, we require as few non-standard features as possible. The exception to this is the DHT that handles account registration, mapping each registered username to a reference in the storage. For resilience against account hijacking, we propose modifying the DHT to be write-once on keys: once an account has been registered, nobody else can register that username.

From the DHT we require two operations, `put`(key, $value$), and `get`(key). The put operation associates the $value$ with the key, and subsequent get operations on that key will return the $value$. As the DHT is write-once, a second put operation with the same key will not affect the system state.

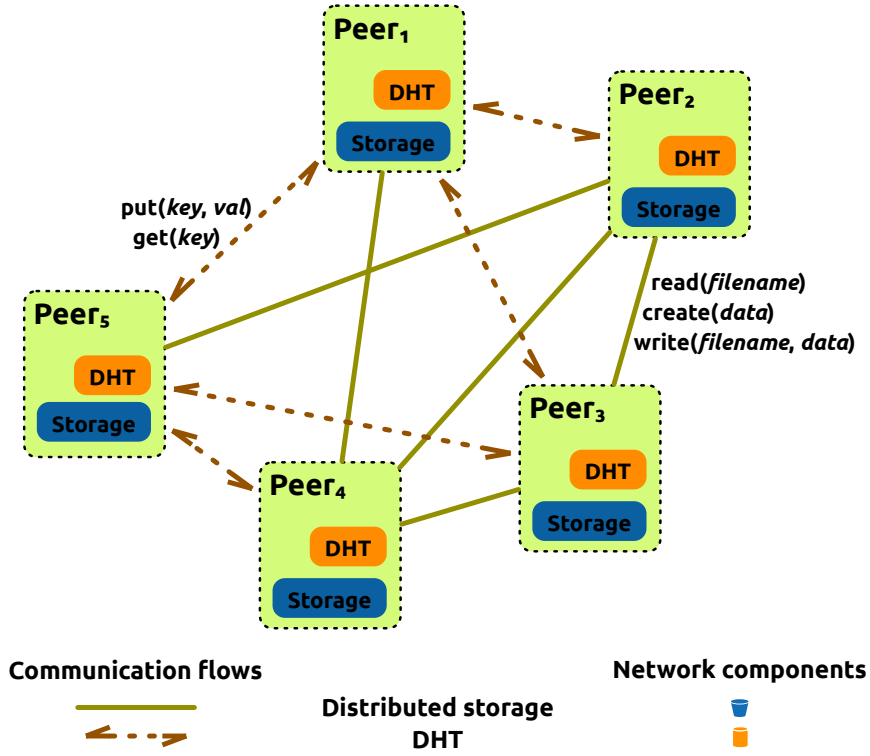


Figure B.1: Overview of the system.

The distributed storage functions for data manipulation are similar to the DHT, with three differences: we allow the distributed storage to select the "filename" for us; we require that data can be updated; and we assume (minimalistic) access control when writing. We refer to what is stored in the distributed storage as files, to simplify our description. While the storage component can be implemented as a distributed file system, we emphasize that our requirements are significantly weaker than full file system semantics.

We formalize the API to the storage as having three operations.
First, `create`(*data*) which generates a new file and returns a filename. Second, `write`($filename$, *data*) that overwrites the file $filename$ with content *data*. Third, `read`($filename$) which reads the content from a file. Our security does not require overwritten data to be inaccessible, so a solution similar to GNUnet [19] or Freenet [36] where a new version is stored and pointed to suffice in our protocols.

We require the storage system to support some minimalistic access control. Each stored file has an owner, which is the user who created the file. Only the owner can perform the `write` operation. To authenticate ownership of files, we assume that a public-key cryptographic system is used.

Finally, for the peer sampling component, we require a `getPeer`() method, returning a randomly selected peer, with a distribution close to uniform.

Password-based P2P Login

For password-based authentication in P2P systems, the basic functionality involved is registering an account, and logging in. We also consider password change and remembered logins, allowing a device to store sufficient information to log in later without asking for credentials anew. Following recommendations from the ISO 27002 standard [70], we define the following requirements for our login procedure and add our own (preceded by a star) to account for several devices.

- passwords should neither be stored nor transmitted in clear text
- a user should be able to choose her own passwords and change them
- files with passwords should be stored separately from application data
- ⋆ a user should use the same password to log in from any device
- ⋆ it should not be possible to recover a password by stealing a device with remembered credentials
- ⋆ it should be possible to block access to the account from a stolen device

The standard also defines limitations for password login procedures that our system cannot provide fully due to the lack of rate-limiting possibilities in P2P networks: to limit the number of unsuccessful login attempts and the maximum and minimum time allowed for the login procedure. Adapting a multi-party password hardening

scheme [56] could, in future work, be a way to achieve similar properties in a P2P network. Besides this limitation, our protocols fulfill the requirements as outlined in the standard, and our own added requirements.

We now describe our protocols based on the system model from Section B.3. Figure B.2 shows the information objects and their storage locations, with arrows for the abstract flow of the login procedure, Table B.1 lists the terms used in the algorithms.

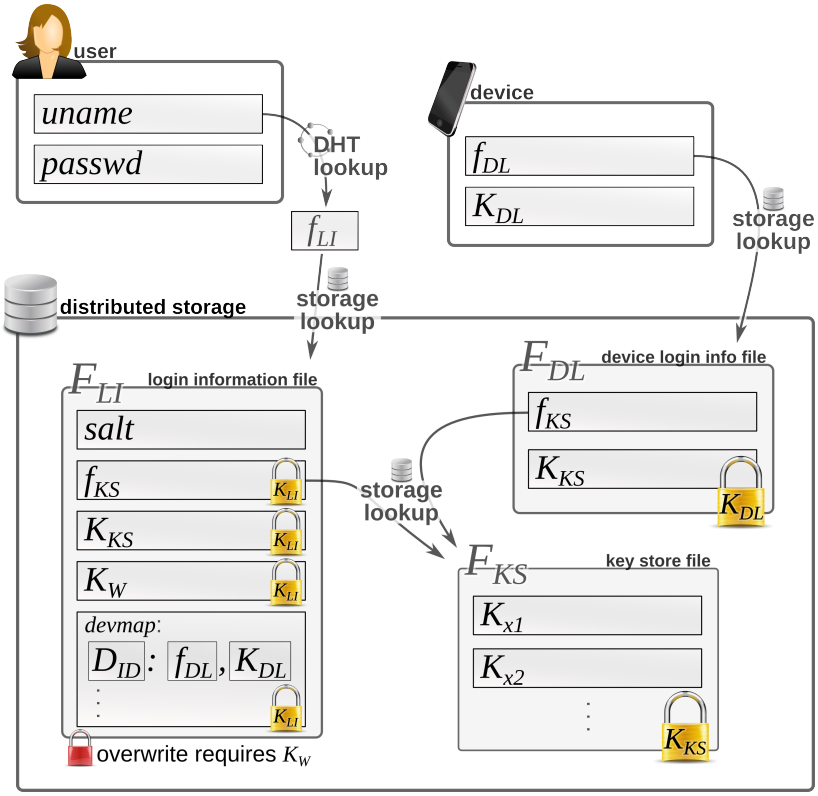


Figure B.2: Storage Locations (boxes) and Login Procedure (arrows)

Account Registration

To register a new account (see Algorithm 1), the user first has to choose a username *uname* and a password *passwd*. Next, the user creates a key store file F_{KS}, containing all the keys used by the P2P application the user wants to log in to (and an additional storage key, authenticating write operations on this file). The user then creates a symmetric key K_{KS}, encrypts the file content with this key

Table B.1: Protocol Terminology

uname	Username
passwd	Password
salt	Random byte string
K_W	Cryptographic key for write authentication
F_{KS}	Key store file
f_{KS}	File name of F_{KS}
K_{KS}	Cryptographic key (used to encrypt F_{KS})
F_{LI}, f_{LI}, K_{LI}	Login information file, its file name and key
F_{DL}, f_{DL}, K_{DL}	Device login information file, its file name and key
D, D_{ID}	User device and the identifier of D
$K_{x1}, K_{x2}, \ldots$	Cryptographic keys for usage after logging in
devmap	Mapping from device identifiers to device login information files and corresponding keys

and puts the ciphertext into the storage, obtaining a file name f_{KS}. Now, the user creates a login information file F_{LI} by creating a random byte string *salt*, deriving a symmetric key K_{LI} from the password *passwd* and the *salt*, encrypting f_{KS}, K_{KS} and K_W (a generated storage key, required for overwriting F_{LI} later) with K_{LI}. The salt and the three encrypted values are put into the storage, obtaining a file name f_{LI}. The salt is stored in plaintext, so that the user later can derive the decryption key K_{LI} by only providing the password. Finally, the user performs the write-once operation `put` on the DHT with *uname* as key and f_{LI} as value. If the username was taken, the user is prompted for a new username. Once all operations have succeeded, the user is registered in the system.

Algorithm 1 Account Registration

1: $uname \leftarrow$ User.input("Choose username:")
2: $passwd \leftarrow$ User.input("Choose strong password:")
3: $K_{KS} \leftarrow$ generateKey()
4: $F_{KS} \leftarrow \text{encrypt}_{K_{KS}}(K_{x1}||K_{x2}||\ldots)$
5: $f_{KS} \leftarrow$ Storage.create(F_{KS})
6: $salt \leftarrow$ generateSalt()
7: $devmap \leftarrow$ createMap()
8: $K_{LI} \leftarrow$ KDF(*salt*,*passwd*)
9: $K_W \leftarrow$ generateKey() // suitable for the storage system
10: $F_{LI} \leftarrow salt||\text{encrypt}_{K_{LI}}(f_{KS}||K_{KS}||K_W||devmap)$
11: $f_{LI} \leftarrow$ Storage.create(F_{LI}) // using K_W
12: **while** DHT.put(*uname*,f_{LI}) fails
13: *uname* $\leftarrow$ User.input("Choose new username:")
14: **end while**

Login

Once registered, a user is able to log in – that is, to retrieve the cryptographic keys stored in the key store file F_{KS} – from any device by only entering her username and password (see Algorithm 2). A `get` request with the parameter *uname* to the DHT results in the filename f_{LI} for the login information file F_{LI}. This file is retrieved from the distributed storage and contains the *salt* in plaintext. The latter is fed into a key-derivation function together with the user password to derive the key K_{LI}. This key allows the user to decrypt all other content of the login information file, including the filename f_{KS} of the key store file and the corresponding key K_{KS}. Finally, the user fetches the key store file F_{KS} from the storage system and decrypts it, using K_{KS}. This concludes the login procedure as the user is now in possession of the keys $K_{x1}, K_{x2}, \ldots$, required by the P2P application.

Algorithm 2 Login

1: $f_{DL}, K_{DL} \leftarrow$ Device.readLocalStore()
2: **if** $f_{DL} \neq$ `NULL` **then** // non-interactive login
3: $F_{DL} \leftarrow$ Storage.read(f_{DL})
4: $f_{KS}, K_{KS} \leftarrow \text{decrypt}_{K_{DL}}(F_{DL})$
5: *saveLoginLocally* $\leftarrow$ `False`
6: **else** // interactive login
7: *uname* $\leftarrow$ User.input("Enter username:")
8: *passwd* $\leftarrow$ User.input("Enter password:")
9: *saveLoginLocally* $\leftarrow$ User.input("Remember?")
10: $f_{LI} \leftarrow$ DHT.get(*uname*)
11: $F_{LI} \leftarrow$ Storage.read(f_{LI})
12: *salt* $\leftarrow F_{LI}.salt$ // stored in plaintext
13: $K_{LI} \leftarrow$ KDF(*salt*,*passwd*)
14: $f_{KS}, K_{KS}, K_W, devmap \leftarrow \text{decrypt}_{K_{LI}}(F_{LI})$
15: **end if**
16: $F_{KS} \leftarrow$ Storage.read(f_{KS})
17: $K_{x1}, K_{x2}, \cdots \leftarrow \text{decrypt}_{K_{KS}}(F_{KS})$
18: **if** *saveLoginLocally* **then**
19: $K_{DL} \leftarrow$ generateKey()
20: $F_{DL} \leftarrow \text{encrypt}_{K_{DL}}(f_{KS}||K_{KS})$
21: $f_{DL} \leftarrow$ Storage.create(F_{DL})
22: Device.writeLocalStore($f_{DL}||K_{DL}$)
23: *devmap*.append(Device.ID, $f_{DL}||K_{DL}$)
24: $F_{LI} \leftarrow salt||\text{encrypt}_{K_{LI}}(f_{KS}||K_{KS}||K_W||devmap)$
25: Storage.write(f_{LI},F_{LI}) // using K_W
26: **end if**

If the user chose to remember the login information on the local device, a new device login information file F_{DL} is created and saved to the storage system (which

returns a filename f_{DL}). This file contains the filename f_{KS} of the key store file as well as the according key K_{KS} and is encrypted with a new key K_{DL}. On the device, only the filename f_{DL} and the key K_{DL} are stored locally. Additionally, a reference to the device login information file is stored in the *devmap* value of the login information file F_{LI}. It contains a mapping from a device identifier to the filename and key of the device login information file, allowing password changes and device revocation as described later.

When the user wants to log in from the same device again, the locally stored values (f_{DL}, K_{DL}) are used to retrieve the device login information file, decrypt it, and thereby gain access to the key store file. Thus, the remembered login feature allows the user to log in without entering the password, while nothing password-related is stored on the device. Furthermore, remembered logins remain valid even if the P2P application changes keys in the key store file.

Password Change

Before the user can change the password, she must log in using her password to obtain K_{LI}. With this information, the password change can be accomplished (see Algorithm 3): the user is asked for a new password and a new salt is generated. The key-derivation function is used to generate a new key K_{LI}^{new} for the login information file. Then, the content of the key-store file is fetched and decrypted (with the old key). A new key K_{KS}^{new} is generated and used for encrypting the key-store content again before it is saved to the storage system, obtaining a new filename f_{KS}^{new}. Finally, the login information file is updated: $f_{KS}^{new}, K_{KS}^{new}$, the write credential K_W as well as a new empty device mapping $devmap^{new}$ are encrypted with the new key K_{LI}^{new}. Together with the new salt, this ciphertext is written to the distributed storage, using the reference f_{LI} and the credential K_W, to authenticate the `write` operation. Lastly, the keys stored in the key store should be updated by the application using our P2P protocol. See Section B.6 for a discussion. At this point, old device login information files can also be deleted from the storage to reclaim space.

Logout

To log out from the system, the user does not have to interact with the DHT or the storage system. Simply wiping her local cache from application data and all key material restores the pre-login state. If the user chose to remember the login on a device, the corresponding device login information file F_{DL} can also be deleted from the storage.

A problem related to logging out is revoking remembered credentials on another device, e. g., a user's stolen phone. To accomplish this, we first run the password change operation, which locks out all devices with remembered logins, because the key store key K_{KS} changed (as well as the filename f_{KS}). Next, we use the device mapping *devmap* to inform all devices about the new key (and filename), except

Algorithm 3 Password Change

Input: $uname, K_{LI}^{old}$

1: $f_{LI} \leftarrow$ DHT.get($uname$)
2: $F_{LI}^{old} \leftarrow$ Storage.read(f_{LI})
3: $f_{KS}^{old}, K_{KS}^{old}, K_W, devmap^{old} \leftarrow \text{decrypt}_{K_{LI}^{old}}(F_{LI}^{old})$
4: $passwd^{new} \leftarrow$ User.input("Enter new password:")
5: $salt^{new} \leftarrow$ generateSalt()
6: $K_{LI}^{new} \leftarrow$ KDF($salt^{new}$,$passwd^{new}$)
7: $devmap^{new} \leftarrow$ createMap()
8: $F_{KS}^{enc-old} \leftarrow$ Storage.read(f_{KS}^{old})
9: $F_{KS} \leftarrow \text{decrypt}_{K_{KS}^{old}}(F_{KS}^{enc-old})$
10: $K_{KS}^{new} \leftarrow$ generateKey()
11: $F_{KS}^{enc-new} \leftarrow \text{encrypt}_{K_{KS}^{new}}(F_{KS})$
12: $f_{KS}^{new} \leftarrow$ Storage.create($F_{KS}^{enc-new}$)
13: $F_{LI}^{new} \leftarrow$
$salt^{new}$|| $\text{encrypt}_{K_{LI}^{new}}(f_{KS}^{new}||K_{KS}^{new}||K_W||devmap^{new})$
14: Storage.write(f_{LI},F_{LI}^{new}) // using K_W
15: Refresh keys stored in key store
16: Old device login information files may be deleted

the device that is to be revoked. To inform a device about the change, we update the corresponding values in the device's login information file F_{DL} which can be accessed from the device by using the locally stored credentials.

Algorithm 4 describes this necessary extension. After running the password change operation, all devices that should *not* be revoked and that have remembered logins (and therefore are referenced in the device mapping *devmap*) are processed. The device login information filename f_{DL} and its key K_{DL} are read, and the new key store key K_{KS}^{new} and filename f_{KS}^{new} are written to the device login information file F_{DL}, encrypted under the device key K_{DL}. Finally, the modified *devmap* is saved back to the login information file F_{LI}.

Algorithm 4 Logout Other Device

1: ... // run Algorithm 3 (Password Change)
2: $deviceToLogout \leftarrow$ User.input("Select device:")
3: $devmap.remove(deviceToLogout)$
4: **foreach** D_{ID} **in** *devmap* // all devices to keep
5: $f_{DL}, K_{DL} \leftarrow devmap$.get($D_{ID}$)
6: $F_{DL} \leftarrow \text{encrypt}_{K_{DL}}(f_{KS}^{new}||K_{KS}^{new})$
7: Storage.write(f_{DL},F_{DL})
8: **end**
9: ... // save modified *devmap* back to F_{LI}

Password Recovery

An important part of password-based logins is the possibility for users to recover their accounts if they forget their passwords. We refer to this as a *password recovery mechanism*. The goal of a password recovery mechanism is to provide a secondary way of authenticating the user. There are a number of password recovery mechanisms used in practice. In our experience, three of the most common ones are password hints, security questions, and e-mail based recovery. Other approaches (beyond the scope of this paper) include vouching for identity by social contacts [25], or using trusted devices.

Password hints means that the user may enter a hint at the same time as she sets this password. The hint will be displayed to her if she forgets her password, and should be selected such that it helps her recall her password, but does not make it significantly easier for someone else to guess it. The hint is not truly a secondary authentication mechanism, but rather a means to recovering the original password-based authentication mechanism. A basic version of password hints would be straightforward to implement in our system: the hint can be stored in plaintext in the login information file. Security questions and e-mail based password recovery are more complex to adapt. We described their implementation in detail after listing requirements.

As in Section B.4 for the login procedure, we define a set of functional requirements for password recovery, based on the ISO 27002 standard [70] as follows. We also augment the list with requirements of our own (preceded by a star).

- establish methods to verify the identity of a user prior to allowing the user to choose a new password
- communicate with those affected by or involved with recovery security incidents
- have procedures to allow recovery and restoration of business operations and availability of information in a time-scaled manner
- a legitimate user should be able to recover lost (forgotten) or broken (device's) keys
- ⋆ the recovery procedure should allow a user to set a new password, not reveal the old password
- ⋆ the process of recovery should be easy to use
- ⋆ sensitive information for recovery should be kept secret

Our protocols support these requirements. The sole exception is that if a password is reset via security questions alone, the system would not "communicate with those affected" (e.g., send an e-mail notification that the password had been reset,

as is common in centralized services). We remark that the last item is a stronger property than many centralized systems provide. In our system, no one learns the answers to a user's security questions. We consider this to be important, since many systems use similar security questions.

The operations described in this section imply minor additions to the protocols of Section B.4, i. e., invoking the update procedures after each password change (to sustain transaction safety, the updates have to be included in the final write operation of the password change operation).

Table B.2: Recovery Protocol Terminology

Security Question Recovery	
qS_i	(n, k)-secret sharing share of K_{LI}
Q_i	Security challenge question
A_i	Answer to question Q_i
$qsalt_i$	Random byte string
qK_i	Key to encrypt the share qS_i
E-mail Based Recovery	
K_R	Long-term recovery key
eS_i	(n, k)-secret sharing share of K_R
$email$	Recovery e-mail address of the user
$peer_i$	Randomly selected peer
$esalt_i$	Random byte string (to seed the e-mail commitment)
$ksalt_i$	Random byte string (to seed the key eK_i)
C_i	Cryptographic commitment to the e-mail address
eK_i	Key to encrypt the share eS_i

Security Questions

Security questions is a password recovery technique that relies on answers to questions the user is asked during registration. The answers should be such that they cannot be easily guessed or researched by an attacker, but still stable over time, memorable, and definite [119]. Rabkin [108] underlines the importance to choose good questions especially in the era of social networks. Frykholm and Juels [58] discuss a related technique that is similar to our adaption of this scheme.

We assume that the user provides n answers A_i to suitable security questions Q_i. In order to recover the password, we require the user to answer any k out of these n questions correctly. The choice of k constitutes an obvious trade-off between security and usability. A successful recovery yields the key K_{LI} to the login information file, allowing the user to change the password, using Algorithm 3. Our implementation does not require the user to provide new answers after a reg-

ular password change. Additionally, we avoid storing the plaintext answers to the security questions.

For the setup of the question based recovery mechanism (Algorithm 5), we first create n shares $qS_1, \ldots, qS_n$ of the key K_{LI} under an (n, k)-secret sharing scheme. For each of these shares, we create a salt $qsalt_i$, derive a key qK_i from this salt and the answer A_i, and use it to encrypt the share, yielding qS_i^{enc}. Furthermore we encrypt the key qK_i with the login information file key K_{LI}, for the update procedure described later. Finally, the login information file is extended with all questions Q_i, the salts $qsalt_i$, the encrypted shares qS_i^{enc} and the encrypted keys qK_i^{enc}. When recovering, the user has to reproduce at least k answers, which together with the stored salts can be used to derive k keys qK_i, which in turn can decrypt k shares qS_i.

When K_{LI} changes (e. g., due to a regular password change), we update the recovery information as in Algorithm 6: for the new key K_{LI}^{new}, a new set of shares is created. Next, the keys qK_i are decrypted and used to encrypt the new shares. Neither the keys qK_i nor the salts $salt_i$ change, so the user can still use the same answers for recovery. Finally, the updated shares (and re-encrypted keys, to allow further updates) are saved back to the login information file.

Algorithm 5 Security Questions Setup

$qS_1, \ldots, qS_n \leftarrow$ createShares(n,k,K_{LI})
for $i \leftarrow 1, n$ **do**
    $Q_i \leftarrow$ User.input("Enter question i:")
    $A_i \leftarrow$ User.input("Enter answer i:")
    $qsalt_i \leftarrow$ generateSalt()
    $qK_i \leftarrow$ KDF($qsalt_i$,A_i)
    $qS_i^{enc} \leftarrow \text{encrypt}_{qK_i}(qS_i)$
    $qK_i^{enc} \leftarrow \text{encrypt}_{K_{LI}}(qK_i)$
end for
add to F_{LI}: qS_i^{enc}, qK_i^{enc} and the plaintext values of $Q_i, qsalt_i \quad \forall i \in \{1, \ldots, n\}$

Algorithm 6 Security Questions Update (on K_{LI} change)

$qS_1^{new}, \ldots, qS_n^{new} \leftarrow$ createShares(n,k,K_{LI}^{new})
for $i \leftarrow 1, n$ **do**
    $qK_i \leftarrow \text{decrypt}_{K_{LI}}(qK_i^{enc})$
    $qS_i^{new-enc} \leftarrow \text{encrypt}_{qK_i}(qS_i^{new})$
    $qK_i^{new-enc} \leftarrow \text{encrypt}_{K_{LI}^{new}}(qK_i)$
end for
update in F_{LI}: $qS_i^{new-enc}, qK_i^{new-enc} \quad \forall i \in \{1, \ldots, n\}$

E-mail Based

In e-mail based password recovery, the user is sent an e-mail containing some information, typically a link with a token, by which she can reset her password. This link is sent to an e-mail address she has registered with her account.

We adapt this scheme by randomly choosing a number of peers, that collaboratively provide this functionality to the user. We use (n, k)-secret sharing to enable password recovery even if not all of the involved peers are online when the user wants to recover the password. We discuss parameter choices of k and n in Section B.7.

To provide persistence of the recovery mechanism independent of a changing key K_{LI} (e. g., due to a password change), the result of the recovery process is a recovery key K_R, that always encrypts the current version of K_{LI}. Algorithm 7 describes the setup procedure: From the recovery key K_R, n shares $eS_1, \ldots, eS_n$ are generated using (n, k)-secret sharing. For each share, a random peer $peer_i$ is picked, two salts $esalt_i$ and $ksalt_i$ are created and a cryptographic commitment C_i is derived from the salt $esalt_i$ together with the *email*. This commitment will be used to authorize the user to the peer, and bind it to this specific e-mail address. Next, a key eK_i, to encrypt the share eS_i, is derived in the same way as the commitment, but with salt $ksalt_i$. A different salt is needed so that the peer cannot decrypt the share (before learning the address). The commitment and the encrypted share are stored at the peer. The login information F_{LI} file is extended with a list of the chosen peers $peer_i$ and the according salts $esalt_i$, $ksalt_i$, as well as K_{LI}^{enc}, encrypted with the recovery key, and the recovery key, encrypted with K_{LI} (to allow for password changes).

Algorithm 7 E-mail Recovery Setup

$K_R \leftarrow$ generateKey() // long-term recovery key
$K_{LI}^{enc} \leftarrow \text{encrypt}_{K_R}(K_{LI})$
$eS_1, \ldots, eS_n \leftarrow$ createShares(n,k,K_R)
$email \leftarrow$ User.input("Enter recovery e-mail address:")
for $i \leftarrow 1, n$ **do**
    $peer_i \leftarrow$ getPeer()
    $esalt_i \leftarrow$ generateSalt()
    $ksalt_i \leftarrow$ generateSalt()
    $C_i \leftarrow$ KDF($esalt_i$,$email$) // commitment
    $eK_i \leftarrow$ KDF($ksalt_i$,$email$)
    $eS_i^{enc} \leftarrow \text{encrypt}_{eK_i}(eS_i)$
    store at $peer_i$: C_i, eS_i^{enc}
end for
$K_R^{enc} \leftarrow \text{encrypt}_{K_{LI}}(K_R)$
add to F_{LI}: K_{LI}^{enc}, K_R^{enc} and the plaintext values of $peer_i, esalt_i, ksalt_i \quad \forall i \in \{1, \ldots, n\}$

To recover the password (Algorithm 8), the user looks up the available information in the login information file, including the list of peers to be requested for assistance. Each request is authorized by the commitment C_i, that the peer can derive from the salt $esalt_i$ and the e-mail address, that the user provided (Algorithm 9). If the request was legitimate, the peer sends the encrypted share to the e-mail address. As soon as the user collected k answers, she can recover K_{LI}.

Algorithm 8 E-mail Recovery: User

1: $uname \leftarrow$ User.input("Enter username:")
2: $email \leftarrow$ User.input("Enter e-mail:")
3: $f_{LI} \leftarrow$ DHT.get($uname$)
4: $F_{LI} \leftarrow$ Storage.read(f_{LI})
5: $K_{LI}^{enc}; \forall i : peer_i, esalt_i, ksalt_i \leftarrow F_{LI}$ // plaintext part
6: $\forall i$: send $(email, esalt_i)$ to $peer_i$ // send n requests
7: $eS_1^{enc}, \ldots, eS_k^{enc} \leftarrow$ read e-mail // wait for k e-mails
8: **for** $i \leftarrow 1, k$ **do**
9: $eK_i \leftarrow$ KDF($ksalt_i$,$email$)
10: $eS_i \leftarrow \text{decrypt}_{eK_i}(eS_i^{enc})$
11: **end for**
12: $K_R \leftarrow$ useShares$(eS_1, \ldots, eS_k)$
13: $K_{LI} \leftarrow \text{decrypt}_{K_R}(K_{LI}^{enc})$
14: ... // run Algorithm 3 (Password Change)

Algorithm 9 E-mail Recovery: Peer

Stored: C_i, eS_i^{enc} // stored at peer
Input: $email, esalt_i$ // provided by the user request
1: **if** $C_i =$ KDF($esalt_i$,$email$) **then** // legitimate request
2: sendMail($email$,eS_i^{enc})
3: **end if**

To provide long-term persistence of this recovery mechanism, K_{LI}^{enc} has to be updated whenever K_{LI} changes. Algorithm 10 describes the necessary steps, including updating K_R^{enc} to allow subsequent updates.

Algorithm 10 E-mail Recovery Update (on K_{LI} change)

1: $K_R \leftarrow \text{decrypt}_{K_{LI}^{old}}(K_R^{enc})$
2: $K_{LI}^{enc-new} \leftarrow \text{encrypt}_{K_R}(K_{LI}^{new})$
3: $K_R^{enc-new} \leftarrow \text{encrypt}_{K_{LI}^{new}}(K_R)$
4: update in F_{LI}: $K_{LI}^{enc-new}, K_R^{enc-new}$

Combining Approaches

The approaches presented above can be composed, either sequentially or in parallel. By sequential composition, we mean that the user must *both* correctly answer security questions and receive e-mail. By parallel composition, we mean that either mechanism can be used alone to recover the password. The latter is achieved by using both systems in parallel.

For sequential composition, the user picks a uniformly random string r of the same length as K_{LI}. The user stores r in one of the mechanisms, and $K_{LI} \oplus r$ in the second mechanism, where $\oplus$ denotes the exclusive-OR operation. If one recovers both of these, K_{LI} can be computed. If one learns only one of the pieces, nothing is gained, as both r and $K_{LI} \oplus r$ are uniformly random. More generally, to combine n mechanisms in arbitrary ways, (n, k)-secret sharing can be used. What we describe here are two trivial such schemes for $n = 2$.

Security

The goal we set is to emulate the security provided by a centralized solution. Some security risks are inherent to the password functionality, and apply regardless of implementation technique. For instance, in e-mail based recovery, an attacker compromising the victim's e-mail account can reset her password.

In this section, we elaborate on security concerns of our protocols. We do not have full cryptographic security proofs of our protocols, something which is important future work.

Concerning safety, we have designed our protocols such that persistently stored data remains in a consistent state if the protocol is aborted at any point. Some protocols may, if an operation fails, leave orphan files in the storage. If our protocol for revoking remembered credentials is interrupted, it may revoke more devices than intended. Apart from this, our operations have transactional semantics, assuming small writes (both creation, and updates) to the storage are atomic operations and that operations block until successful.

Adversary Model

To capture the concept of collusions, we consider an adversary that corrupts a number of nodes. Upon corruption, the adversary gains all information known to that node, and in the case of an active adversary, can also control its future actions. The adversary can also make requests to the underlying system, e. g., read files from the distributed storage. As almost all our protocols mainly operate on publicly readable (encrypted) data, this ability is important. The only computation made by nodes different from the one logging in are in verifying write operations, and in e-mail based password recovery.

Risks in Used Components

As our protocols make use of several standard components, vulnerabilities in those components can also affect our system. For instance, an adversary may prevent a user from logging in by attacking the victim's ability to read her login information file from the distributed storage. We note that there are many security techniques in DHTs that can mitigate such threats, and refer to Urdaneta et al. [129] for a recent survey.

Offline Guessing

An issue in password authentication based on a (distributed) file system is that the information required to verify a password attempt is inherently exposed. This means that in our scheme, we cannot prevent an attacker from mounting an offline attack against our encrypted passwords. This is a considerable drawback from centralized schemes, where the encrypted password database is kept protected.

As a partial mitigation, we utilize a KDF with a per-user salt. This forces an attacker to evaluate the KDF individually for each user on a password guess, defeating parallel attacks against multiple users. We recommend the system be instantiated with a slow KDF, such as bcrypt [107] to throttle offline guessing. The protocol could be modified to reduce storage by using the username as a salt, but we recommend against that as it would be vulnerable to pre-computation (before the system is started, or between instances using the same KDF) attacks against common username-password pairs.

The problem of a server performing offline attacks against its password database was treated by Ford and Kaliski [56]. Their techniques are client-server based, and require all servers to be online for a login. We leave it as future work to investigate modifying their protocol to be applicable also in a P2P setting. This would prevent offline guessing attacks.

Colluding Nodes in E-mail Based Recovery

In the mechanism for e-mail based recovery, we employ (n, k)-secret sharing, and secrets are stored on n random nodes in the system. If an attacker controls k or more of these nodes, she can recover the secret and access the victim's account. In Section B.7 we discuss the choice of these parameters.

A peer sampling protocol is used to select the n nodes where the shares are stored. An active attacker may influence this protocol, in order to ensure that she controls k out of the selected nodes. This can be mitigated by a peer sampling protocol designed to tolerate active attacks, such as Brahms [23].

The protocol is designed to reveal only minimal information to the peers. Thus, even when colluding, peers have to get hold of both, the salt $ksalt_i$ and the recovery e-mail address *email* to mount an attack. The e-mail address will be revealed to a peer only when the user initiates the recovery process. Malicious peers might try to

guess it earlier, but to verify guesses either $ksalt_i$ or $esalt_i$ are required. These are stored in the login information file of the user, which can only be found knowing the username. Therefore the setup process should be anonymous, where the peer does not learn the username of the user for whom it stores the share.

Updating Application Keys

When a password is changed, or device is revoked, access is effectively revoked from future updates to the key store. However, a malicious device may have stored the last keys it was able to access. Thus, in the password update procedure (which is also used for device revocation), the keys for the P2P application itself need to be updated, and then these new keys need to be written to the key store. How to update them, if possible, is beyond the scope of this paper, as it is a functionality of the application protocol.

Denial-of-Service

An adversary may mount a DoS attack by writing many user names into the DHT, thus blocking those from registration by legitimate users. This attack is also possible in centralized systems, but there detection and counter-measures (e.g., removing the fake accounts) is significantly easier. One can solve this issue by assuming a lightweight CA dealing only with checking user identification before account creation, similar to Safebook [39]. We remark that this attack only target the availability of registration of a new user account, it does not affect existing users.

A related attack can occur when an adversary can prevent a user from accessing some of the data required to log in (e.g., by controlling all replicas holding the victim's entry in the DHT). In such attacks, existing users can be prevented from logging in, possibly permanently by overwriting or deleting the key. This illustrates the importance of applying security techniques for underlying components [129] and indicates a large replication factor should be chosen.

Security Summary

Aside from the concerns outlined, we believe that our schemes produce a similar level of security as client-server based password authentication schemes. The cryptographic design of our protocols relies on relatively standard techniques. This leads us to be confident that the security of our protocols can be formally proven using cryptographic techniques.

We also provide some features which are not commonly present in centralized systems. One of these is the ability to revoke stored credentials from only some specific devices. A second one is the ability to set up e-mail based password recovery without revealing your e-mail address before recovery actually occurs, offering an additional privacy protection.

Evaluation

We developed two lightweight custom simulators, one to evaluate the efficiency and security of our protocols, and one to assist in setting the n and k parameters for e-mail based password recovery. For the performance analysis, we take as input the time to perform required cryptographic operations, as well as the time of our network operations. For the analysis of n and k parameters, we need data on node uptime to evaluate the availability of the password recovery system for a given choice. We used latency data from Jiménez et al. [74], and node availability data from Rzadca et al. [116].

To measure the computational cost of the necessary cryptographic operations in a prototype implementation, we used OpenSSL's built-in benchmark function on a 2.26 GHz Core 2 Duo running Mac OS X, as well as an ARM 1 GHz Cortex-A8, similar to modern smart phones. The times for all required cryptographic operations (using DSA as a public-key scheme) was negligible, below 5 ms (2 ms on the faster CPU).

As our protocols can be applied with any DHT and file storage combination, we used the BitTorrent Mainline DHT as an example for our numeric performance evaluation. Jiménez et al. [74] recently ran experiments to evaluate the performance of their proposed algorithmic improvements. From their measurements, we received a CDF for the latency of real-world DHT lookups. We assume that all our *network operations*, writes and reads, both from DHT and distributed storage, take the same amount of time as a DHT lookup in their study. This can be motivated, as our distributed storage could be implemented via a DHT. We remark that their measurements are performed with a "warmed up" client with filled routing tables for the DHT. Thus, these numbers may be overly optimistic for a newly started client.

Finally, we believe, node availability will vary considerably between applications. As a representative case, we considered a distributed storage system by Rzadca et al. [116], which featured such data in their evaluation. In the distribution, 10% of nodes have availability 95%, 25% have availability 87%, 30% have availability 75%, and 35% have availability 33%. To this rough distribution, Gaussian noise with $\sigma = .1$ is added, and the resulting availability is capped between 3% and 97%.

Performance

We believe that the main performance-critical operation is logging in [115]. We believe that for all other operations, latency on the order of a few seconds can be acceptable, and even a minute if they are run in the background. Thus, we only present results for logging in, but note that as the operations for other protocols are similar, results are expected to be similar. The performance cost of our protocols is dominated by network operations. However, to slow down password guessing attempts, one may wish to force the key derivation procedure to be slow, to the point of making that cost dominant.

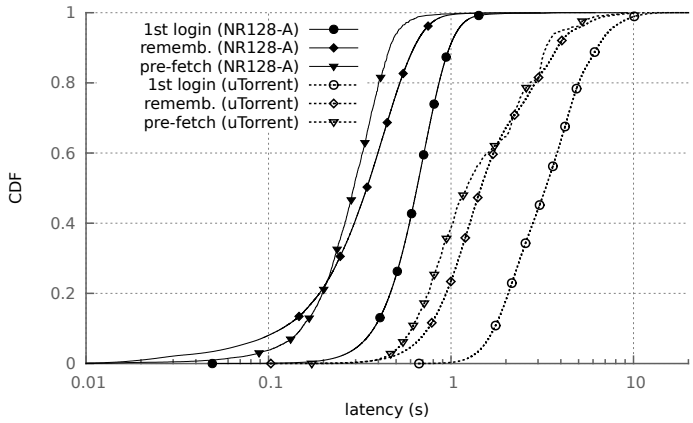


Figure B.3: CDF for login latency in three modes: First time login, remembered logins, and first time login after password entry (pre-fetch). Network operations are assumed to have costs of BitTorrent mainline DHT lookups, using NR128-A (solid lines) or μTorrent strategy (dashed lines) [74].

Table B.3: Latencies of protocols, in milliseconds.

	Network Op. [74]		First Login		Remem. Login	
DHT	median	99^{th}	median	99^{th}	median	99^{th}
NR128-A	164	567	650	1362	346	915
μTorrent	647	5140	3299	10154	1456	7148

When evaluating the protocols, we parallelized network operations where possible. Logging in for the first time and remembering the credentials for future logins is then a sequence of two network operations, followed by key derivation, followed by three parallel network operations. In a password login, it is also possible to pre-fetch some of the information after the user has entered her username, but before she enters her password. In particular, as soon as the username is known, the filename F_{LI} can be retrieved from the DHT, and the file can be read. Decryption of the file and further processing is then only possible after the user enters her password. To evaluate this speed-up, we computed the time it takes to finish the login after the user has entered her password. The time to fetch the two files is identical to the time to do a remembered login, and it is sufficiently small that the data can realistically be retrieved while the user is typing her password.

To determine the sensitivity of our performance to implementation characteristics, we evaluated our protocol for two different client strategies in the BitTorrent Mainline DHT: The NR128-A algorithm [74], and the μTorrent client's implementation. We present these performance numbers in Figure B.3 and Table B.3. Firstly,

we observe that with a fast storage, our login protocol is very fast, with a median login time of 650 ms the first time, and 346 ms for remembered logins. Comparing the results, we observe that our protocols are indeed sensitive to storage latency. While performance results building on μTorrent data are slightly above recommended levels [127], we still consider them within range of acceptability for P2P.

When evaluating these numbers, we assumed that the run-time of the KDF function is negligible. As a system designer, one may wish to pick a slow KDF (e. g., bcrypt [107]), as this slows down password guessing attempts. Any latency intentionally added via the KDF would affect the first-time login (after the password entry) times.

Parameters for E-mail Password Recovery

There are trade-offs between availability, security, and storage space in our e-mail based password recovery protocol.

For the selection of k, the minimum number of peers required to recover the password, there is a direct trade-off between security and availability. Lower choices of k increase the risk of an adversary, controlling a significant number of nodes, to break into the user's account. A higher k reduces the availability of the recovery functionality, which reflects the chances of a user to immediately succeed with the password recovery. However, if the user does not instantly receive k answers, she can simply wait until enough peers are online.

We believe a reasonable choice of parameters is $n = 16$ and $k = n/2$. With these numbers, using the availability data from Rzadca et al. [116], there is a 96% probability of immediate recovery of a lost password. A very strong attacker, corrupting 25% of the nodes in the system, would still only be able to access the user's account with probability 3%. The analysis of parameter choice here is similar to any P2P system using secret sharing, and we refer to e.g., Vu et al. [132] for a more in-depth discussion.

Scalability

The latency of our protocols will scale similarly to DHTs or other distributed storage systems. The data we used for evaluation is based on measurements on the largest deployed DHT, demonstrating that performance is good with extremely large user numbers. Performance may in fact be worse for a small system, as there are then fewer nodes, meaning that it is less likely to find data at a nearby node. To bootstrap the system with good performance when it is small, a very simple distributed storage using one or a few super-nodes would be one approach. Storage requirements per user are also small, with a few files per user and small file sizes. From this, we conclude that our system is likely to scale well with the number of users.

Conclusions and Future Work

The pros and cons of password-based authentication have been extensively debated. We believe that for some applications, a username-password pair provides an appropriate level of security. We argue that incorporating a well-known authentication scheme may assist in user adoption of P2P systems for more complex tasks than file sharing. To the best of our knowledge, ours is the first work to focus on password-based logins in a P2P setting, including mechanisms to recover a forgotten password. Our protocols are new (but our security questions are similar to [58]), relatively straightforward, and we believe, they are an important first step towards usable authentication in P2P.

The performance of our mechanisms in terms of delay varies according to the underlying DHT or P2P system in general and in relation to how much intentional delay is added by parameterizing cryptographic functions. Overall, however, our evaluation results show that for user satisfaction [115], the delays can be kept at a very acceptable level [127].

While we have provided an initial discussion of the security properties of our protocol here, future work will include a thorough security analysis. Our scheme allows offline password guessing attacks, which will also be addressed in future work.

Acknowledgments

We thank Raúl Jiménez et al. [74] for sharing their measurement results and Jay Lorch for excellent work as a shepherd of this paper. This research has been funded by the Swedish Foundation for Strategic Research grant SSF FFL09-0086 and the Swedish Research Council grant VR 2009-3793.

Article C

User Search with Knowledge Thresholds in Decentralized Online Social Networks

C

Benjamin Greschbach, Gunnar Kreitz, and Sonja Buchegger

KTH Royal Institute of Technology
School of Computer Science and Communication
Stockholm, Sweden
{bgre, gkreitz, buc}@csc.kth.se

Abstract

User search is one fundamental functionality of an OSN. When building privacy-preserving DOSNs, the challenge of protecting user data and making users findable at the same time has to be met. We propose a user-defined knowledge threshold ("find me if you know enough about me") to balance the two requirements. We present and discuss protocols for this purpose that do not make use of any centralized component. An evaluation using real world data suggests that there is a promising compromise with good user performance and high adversary costs.

Introduction

Popular Online Social Networks (OSNs) are logically centralized systems. The massive information aggregation at the central provider inherently threatens user-privacy. Data leakages, whether intentional (e. g., selling of user data to third parties) or unintentional (e. g., by attacks from outsiders), happen regularly[1]. Mo-

[1]To name only two examples: Twitter leaking data from 250K users in February 2013 (`http://blog.twitter.com/2013/02/keeping-our-users-secure.html`), Facebook selling user data (`http://www.telegraph.co.uk/technology/facebook/8917836/Facebook-faces-EU-curbs-on-selling-users-interests-to-advertisers.html`).

tivated by this insight, Decentralized Online Social Networks (DOSNs) have been proposed to mitigate the threats. When decentralizing a system, two challenges have to be met: to implement equal functionality without centralized components, and to provide user privacy under a significantly different threat model.

Here, we look at the functionality of user search, i. e., the lookup of a system-specific user identifier (e. g., a URI of a profile) based on information about the user (e. g., name, city, affiliation). The ability to search for users, in conjunction with other ways of traversing the social graph (e. g., friendlist of friends), is a basic building block of an OSN that allows users to find each other and thereby establish links.

Our Contribution

We propose and evaluate protocols to support user search in a decentralized OSN that shield user data from parties who know less than a user-specified threshold amount of information about the target. To our knowledge, formalizing the use of this consideration is a novel application of knowledge-based access control. This type of restriction was inspired by an observation by Fong et al. [55] that being able to reach a user in an OSN is an integral part of access control in such systems.

We evaluate our protocols using real world data from the U.S. census to relate the performance for legitimate users to the costs of an adversary attempting to guess unknown information.

Related Work

To the best of our knowledge the privacy-findability tradeoff has not been formally investigated in this context. The closest example is user search in Skype. However, as far as we know, their protocol has not been described in detail, but only via external measurement studies, such as one by Baset and Schulzrinne [17].

Most user search functionalities, including ours, search for users within the global user database of the OSN, independently of who searches. In contrast, we note that recently, Facebook has debuted Graph search [48], which ties searching to the social graph, and where the goal is not only to find users, but also content. Several other approaches of personalized searching for content in an OSN have also be discussed, e. g., by Bai et al. [15] in a decentralized setting.

Although designed specifically to search for users in a DOSN, some challenges are shared with constructing a general purpose search in a peer-to-peer (P2P) setting. This has been studied by e. g., Li et al. [86], and Bender et al. [18]. There is also a commercial search engine using P2P, Faroo [50]. Two differences are our focus on access control and privacy, and the significantly smaller amount of information to be indexed in our setting. Similar to these proposals, we also build upon a Distributed Hash Table (DHT) as a core component to realize our functionality.

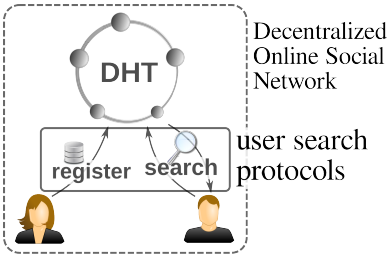


Figure C.1: System overview: The search protocols are one component of the DOSN and makes use of a DHT.

Decentralized User Search Protocol

As we design search protocols for a decentralized system, we cannot assume any trusted third party or central search provider to be available. Instead, we use a DHT to register and look up search terms, as it is a common component of DOSNs. As the DHT runs on nodes participating in the system, we must also protect the privacy of the participants against these nodes.

We propose two protocols, both designed to index and retrieve information in a DHT in a protected way. Our protocols provide two operations. A *register* operation, where users enter information that allows others to find them based on certain attributes, and a *search* operation that, given a set of search terms, returns the set of matching user identifiers. In a next step, out of the scope of the search protocols described here, these user identifiers can be used to view public profiles, and to send a message or friend request to the found user. Figure C.1 illustrates the search functionality.

Protocol Specification

We consider a searcher, who wants to find a searchee. The searchee registers searchable information about herself in the DHT by choosing a number n of attribute labels l_i (e. g., lastname, firstname, city) and assigning each one[2] value v_i. This label-value pair (denoted as attribute a_i) is mapped to a user identifier uid of the searchee. Upon registration the searchee specifies a threshold number t of attributes which the searcher must know in order to obtain the user identifier.

Storing Values in the DHT

The DHT holds a mapping from user attributes to user identifiers, but this mapping must be protected, also against the nodes in the DHT. To this end, we propose

[2]For simplicity we assume that each attribute can be assigned only exactly one value.

a protocol that alters how values are added and retrieved from the DHT. The required property is to retain standard DHT functionality, while nodes in the DHT do not learn plaintexts of keys or values.

When storing a *key-value* pair the *key* is fed into a Key Derivation Function (KDF) together with a global salt $gSalt$, yielding the DHT-key for the `put` and `get` operations of the DHT. The *value* is encrypted using a secret that is derived from a random salt *salt* and the *key* (the attribute information, in our case). The *salt* is stored together with the ciphertext on the right hand side of the mapping. In short, the mapping of a *key-value* pair in the DHT looks like this:

$$\mathrm{KDF}(gSalt,key) \mapsto salt||\mathrm{encrypt}_{\mathrm{KDF}(salt,key)}(value)$$

The $gSalt$ has to be publicly available for all users to allow the lookup of any attributes. This invalidates the purpose of a salt, as pre-computing tables to reverse the left hand side becomes possible again. Nevertheless, we suggest to keep the $gSalt$ as it at least requires the pre-computation attack to be targeted to each specific instance of our system and off-the-shelf pre-computed tables for the used KDF cannot be employed.

The *salt* is an individual random number different for every entry. Note that it in particular has to be different from $gSalt$ as otherwise any DHT node could decrypt the *value* of items it stores, using the left hand side (without knowing the *key*).

Scheme 1: Storing all Allowed Attribute Combinations

We want a searcher to prove knowledge of a threshold number of attributes before obtaining the user identifier. One direct approach to achieve this is to map the user identifier only from attribute concatenations of the threshold length. If the searchee registered e. g., seven attributes and specified that at least four of them are necessary to find her *uid*, we would store the following $\binom{7}{4} = 35$ combinations:
$a_1||a_2||a_3||a_4 \mapsto uid$
$a_1||a_2||a_3||a_5 \mapsto uid$
...
$a_4||a_5||a_6||a_7 \mapsto uid$

where $a_i = (u_i, v_i)$, u_i attribute labels and v_i attribute values. We assume there is a canonical order of attributes (e. g., a lexicographic order of labels), and attributes are sorted by this order before concatenation.

Algorithms 11 and 12 describe the protocol in more detail. For registration, all attribute combinations of length t are mapped to the user identifier and stored in the DHT according to the procedure described in Section C.2. When searching, all provided search attributes are ordered and used to query the DHT (after the Section C.2 transformation). If the result is empty or does not contain what the user was looking for, all subsets of the provided search attributes are subsequently tried,

Algorithm 11 Registration (Scheme 1)

1: $l_1, \ldots, l_n \leftarrow$ User.input("Choose searchable attribute labels (e. g., name,city,...)")
2: $v_1, \ldots, v_n \leftarrow$ User.input("Enter values (your name, your city,...)")
3: $a_i \leftarrow l_i || v_i$ // for $i = 1 \ldots n$
4: $t \leftarrow$ User.input("Enter threshold number of attributes necessary to find you.")
5: **for** all ordered sequences $a_p || \ldots || a_q$ of length t **do**
6: $key \leftarrow a_p || \ldots || a_q$
7: $dhtkey \leftarrow \text{KDF}(gSalt, key)$
8: $salt \leftarrow$ generateSalt()
9: $value \leftarrow uid$
10: $dhtvalue \leftarrow salt || \text{ encrypt}_{\text{KDF}(salt,key)}(value)$
11: DHT.put($dhtkey$,$dhtvalue$)
12: **end for**

ordered by decreasing number of elements. The final result will contain the user identifier of the searchee (and possibly more hits from other users that registered the same attributes) if the number of attributes searched for is greater or equal than the threashold specified by the searchee.

Algorithm 12 Search (Scheme 1)

1: $l_1, \ldots, l_s \leftarrow$ User.input("Choose attribute labels to search for (e. g., name,city,...)")
2: $v_1, \ldots, v_s \leftarrow$ User.input("Enter attribute values (a name, a city,...)")
3: $a_i \leftarrow l_i || v_i$ // for $i = 1 \ldots s$
4: **for** $i \leftarrow s, \ldots, 1$ **do** // while result set is empty or the user requests more results
5: **for** all ordered sequences $a_p || \ldots || a_q$ of length i **do**
6: $key \leftarrow a_p || \ldots || a_q$
7: $dhtkey \leftarrow \text{KDF}(gSalt, key)$
8: **for** $salt, ciphertext$ **in** DHT.get($dhtkey$) **do**
9: $uid \leftarrow \text{decrypt}_{\text{KDF}(salt,key)}(ciphertext)$
10: add uid to result set if decryption was successful
11: **end for**
12: **end for**
13: **end for**

One shortcoming of this scheme is that for sufficiently large numbers of n and t, the number of combinations might become infeasible for storage space constraints and KDF computation latencies during registration. Requiring e. g., 5 out of 20 registered attributes would yield 15504 combinations.

Scheme 2: Storing Each Attribute Individually

An alternative approach, overcoming the large number of combinations generated by Scheme 1, is to store each attribute individually. In order to require a threshold number of attributes to find the user identifier, a single attribute does not map directly to the *uid* but to an encrypted version. The key used for the encryption is based on a secret sharing scheme and one share is stored with each of the attributes. Instead of using the shared key directly, it is fed into a KDF together with an individual salt. This indirection allows us to independently tune the costs for requesting shares for one attribute (determined by the DHT latency and the KDF described in Section C.2) and for trying to combine them (determined by the KDF used here). Furthermore, a bloom filter bf_i is attached to each share, to help finding the right shares to combine with, which is important for popular attributes with large response sets:
$a_1 \mapsto share_1||bf_1||salt_1||\ \text{encrypt}_{\text{KDF}(salt_1,sk)}(\text{uid})$
...
$a_n \mapsto share_n||bf_n||salt_n||\ \text{encrypt}_{\text{KDF}(salt_n,sk)}(\text{uid})$
where sk can be recovered with t of the shares $share_1 \ldots share_n$.

The bloom filter that is stored with each share is created using all other $n-1$ shares belonging to the same key sk. To avoid the case in which two bloom filters for a related set of shares look similar, we introduce an individual salt for each bloom filter, which is used to modify elements before insertion. Thus, with each bloom filter bf_i, we store a salt $bfsalt_i$, and when adding or querying for an element (a share in our case) in bloom filter bf_i, we first hash the element together with the $bfsalt_i$. E. g. instead of bf_i`.add(`$share$`)`, we do bf_i`.add(hash(`$bfsalt_i, share$`))`, where `hash()` is a cryptographically strong keyed hash function.

Algorithms 13 to 16 describe the protocol in more detail. When combining the shares in the search protocol, the bloom filter information is used to reduce the number of possible combinations. Note that for two sets of shares (and attached bloom filters) two reductions are possible: First a share in set one is fixed and its bloom filter is used to reduce set two. Then, for all remaining shares in set two, their bloom filters can be used to determine if they fit to the fixed share of set one. If not, they are removed from set two as well. This generalizes; for n sets, in expectancy the number of matches will be reduced by a factor of $\exp(bloomfactor, \sum_{i\in 1\ldots n} 2(i-1))$, where $bloomfactor$ is the false positive probability of the bloom filter.

Extensions

Weighting of attributes. Some attributes might be easier to guess for an attacker than others because they have a lower entropy or represent more public information that is easy to research from system external sources. We therefore want to give the users the ability to weight attributes, that is, differentiating their contribution for reaching the threshold number t. In Scheme 1, this is straightforward to implement: instead of registering all attribute combinations with a certain

Algorithm 13 Registration (Scheme 2)

1: $l_1, \ldots, l_n \leftarrow$ User.input("Choose searchable attribute labels (e. g., name,city,...)")
2: $v_1, \ldots, v_n \leftarrow$ User.input("Enter values (your name, your city,...)")
3: $a_i \leftarrow l_i || v_i$ // for $i = 1 \ldots n$
4: $t \leftarrow$ User.input("Enter minimum number of attributes necessary to find you.")
5: $sk \leftarrow$ generateKey()
6: $share_1, ..., share_n \leftarrow$ createShares(t,n,sk)
7: **for** $i \leftarrow 1, \ldots, n$ **do**
8: $key \leftarrow a_i$
9: $dhtkey \leftarrow$ KDF(gSalt,key)
10: $bf \leftarrow$ createBloomFilter($\{share_j | j \neq i\}$) // using salted bloom filter
11: $salt \leftarrow$ generateSalt()
12: $k_E, k_S \leftarrow$ KDF($salt$,sk) // derive keys to encrypt and sign
13: $ciphertext \leftarrow \text{encrypt}_{k_E}$(uid)
14: $value \leftarrow share_i || bf || salt || ciphertext || \text{MAC}_{k_S}(ciphertext)$
15: $dhtsalt \leftarrow$ generateSalt()
16: $dhtvalue \leftarrow dhtsalt || \text{encrypt}_{\text{KDF}(dhtsalt,key)}$(value)
17: DHT.put(dhtkey,dhtvalue)
18: **end for**

number of attributes, we only register combinations whose weighted sum meets the threshold.[3] For Scheme 2, more work has to be done, to implement this functionality. A possible approach is, to first pick a granularity number g for the weighting factor (the number of discrete values the weighting factor can take). Instead of storing only one share with each attribute, 1 to g shares will be stored with each attribute depending on the weight for this attribute. The threshold number will be adjusted accordingly (e. g., multiplied by g). To hide the weight of an attribute, all attributes with less than maximum weight will store dummy shares. Following a convention to first store the real shares and than append dummy shares, the additional work (for legitimate users as well as adversaries) – when trying combinations of share values – is guessing this split-point between real and dummy shares for each attribute (e. g., for $g = 10$ and 4 shares, a factor of 10000).

Dummy-attributes for Plausible Deniability. Introducing plausible deniability for leaked personal information can mitigate the consequences of privacy breaches. This can be accomplished by adding random dummy-attributes along with the real attributes. Thus, the adversary cannot be sure if an attribute that she found to be related to a user is a real one or a generated fake entry. Dummy-

[3] More precisely: only those combinations where the weighted sum is greater or equal the threshold and the removal of any included attribute would yield a weighted sum less than the threshold.

Algorithm 14 Search (Scheme 2)

1: $l_1, \ldots, l_s \leftarrow$ User.input("Choose attribute labels to search for (e. g., name,city,...)")
2: $v_1, \ldots, v_s \leftarrow$ User.input("Enter attribute values (a name, a city,...)")
3: $a_i \leftarrow l_i || v_i$ // for $i = 1 \ldots s$
4: $setOfShareSets \leftarrow \emptyset$
5: **for** $i \leftarrow 1, \ldots, s$ **do**
6: $key \leftarrow a_i$
7: $dhtkey \leftarrow$ KDF(gSalt,key)
8: $shareSet \leftarrow \emptyset$
9: **for each** $(dhtSalt, dhtCiphertext) \in$ DHT.get(dhtkey) **do** // more than one result possible
10: $share||bf||salt||uidCiphertext||mac \leftarrow$ $\text{decrypt}_{\text{KDF}(dhtSalt,key)}(dhtCiphertext)$
11: $shareSet$.add($(share, bf)$) // also remember $salt$, $uidCiphertext$, mac
12: **end for**
13: $setOfShareSets$.add($shareSet$)
14: **end for**
15: $sk \leftarrow$ reduceAndCombineShares($setOfShareSets$, $\emptyset$) // recovers sk iff $s \geq t$
16: $salt, uidCiphertext, mac \leftarrow$ lookup values for successful shares // see line 11
17: $k_E, k_S \leftarrow$ KDF($salt$,sk)
18: $uid \leftarrow \text{decrypt}_{k_E}(uidCiphertext)$ // and validate mac using k_S

attributes come, however, with the trade-off of increasing false positive matches for legitimate users. Furthermore, they can be debunked by adversaries with background knowledge. Finally, they might make brute-force attacks easier, as dummy-attributes increase the total number of attributes but not the threshold number of required attributes.

Threat Model

All information that the user gives away or generates while interacting with the system has to be considered as possibly sensitive. This comprises general administrative information (existence in system, date of registration, user-identifiers), entered information during registration (attributes, i. e., label-value pairs), search query data (who searches for whom, which previously unknown attributes are used to specify search-target) and behavioural data (online times, frequency of searching/registering/updating information).

Algorithm 15 reduceAndCombineShares (Scheme 2)

Input: $setOfShareSets, chosenShares$
Output: sk

if $|setOfShareSets| = 0$ **then** // base case: try to recombine candidate shares
  $sk \leftarrow$ useShares($chosenShares$)
  if sk valid **then**
    return sk // for simplicity, return only the first valid key
  end if
  return $None$
else // otherwise recurse
  $S \leftarrow setOfShareSets[0]$
  $SRest \leftarrow setOfShareSets \setminus S$
  for $(share, bf) \in S$ **do**
    $SRestReduced \leftarrow$ reduceShareSets($share, bf, SRest$)
    $result \leftarrow$
      reduceAndCombineShares($SRestReduced, chosenShares||share$)
    if $result \neq None$ **then**
      return $result$
    end if
  end for
  // if nothing was returned yet, try not to pick any share from the current set
  return reduceAndCombineShares($SRest, chosenShares$)
end if

Algorithm 16 reduceShareSets (Scheme 2)

Input: $share, bf, setOfShareSets$
Output: $reducedShareSets$

$reducedShareSets \leftarrow \emptyset$
for $S \in setOfShareSets$ **do**
  for $share', bf' \in S$ **do**
    if not checkBloomFilter($share', bf$) **then**
      S.remove(s')
    end if
    if not checkBloomFilter($share, bf'$) **then**
      S.remove($share'$)
    end if
  end for
  $reducedShareSets$.append(S)
end for
return $reducedShareSets$

Adversaries and Their Capabilities

All agents in the system can possibly act in malicious ways. This comprises nodes involved in the DHT storage, passive traffic observers and active adversaries, i. e., malicious users that can perform search and register operations. Their capabilities range from sniffing traffic and performing traffic analysis (e. g., analyzing query sizes), crawling the DHT (performing massive search operations) or analyzing data they might store, to actively inserting data into the DHT.

Example instances of these adversary models are curious users of the system, targeted attacks from parties with background knowledge about the target user (e. g., testing specific attributes of this user, also learning from negative results), or crawling attacks that aim to harvest information for e. g., spammers, targeted advertisement or insurance companies. We cannot perform a comprehensive security and privacy analysis of the protocols, taking into account all mentioned user assets and adversary capabilities. Instead, we will focus on several specific attacks and present one of them in more detail.

Subset Crawling Attack Scenario

The proposed protocols are trying to balance findability and privacy. Thus, they cannot provide perfect protection. In the worst-case of a targeted attack, an adversary with profound background knowledge about the target user will likely succeed. For example protecting the user identifier cannot be accomplished if the adversary knows as many attributes about the target user as legitimate users do. At the same time we assume that both schemes protect the users fairly well from large-scale crawling attacks as the search space of all possible attribute combinations is too large to brute-force and the protocols transform the registered user data in a way that inferences from the publicly stored data are infeasible. If an adversary chooses to constrain her effort to only crawl the data of a specified subset of the user-base, her chances might be better. We therefore focus on what we call a Subset Crawling Attack. In this scenario, the adversary chooses a number of sensitive attributes and tries to identify all users of the system that registered that attributes. For example, the adversary could try to identify all users working at a specific company by fixing the attribute "workplace" and then brute-forcing a set of identifying attributes such as "name", "firstname" and "city".

Privacy Evaluation

In the following we will evaluate the costs for an adversary to perform a Subset Crawling Attack and compare this to the search costs of legitimate users. We assume that a person's first name, last name, and the city the person is located at are identifying attributes and at the same time among the most popular search attributes. These attributes might be rather public information or easy to research, so we assume that users combine them with other, less public attributes. According

to the Subset Crawling Attack scenario we assume that the adversary fixes at least one of the other attributes and tries to brute-force the identifying attributes.

Data sources

To get evaluation results reflecting realistic distributions of values for identifying attributes, data from the U.S. census was used as input for the following calculations.

Distributions of U.S. *last names* were taken from [143, Table 1]. The data shows that there are 4 Million last names in total. 7 last names occur more than 1 Million times in the U.S. population. The top 3012 last names are shared by 55% of the population, the top 1 Million names are shared by 98.5%. The last name frequencies roughly resemble a power law distribution. Frequencies of U.S. *first names* were taken from [130]. The data is split into "male" and "female" first names. For our calculations we merged them assuming an equal distribution of the two categories. For U.S. *cities*, we used a dataset listing the population of all cities with more than 50000 inhabitants [131]. The data closely resembles a Zipf distribution. For the remaining population, we made a worst-case assumption of being distributed equally to cities with 50000 inhabitants (worst-case in the sense of getting less diversity for this attribute).

The validity of the evaluation results is therefore based on the assumption that the system's user base is a representative subsample of the U.S. population. In the following calculations we furthermore assume that all users registered all three attributes. A source of errors in our evaluation is that we treat these attributes as independent, because we were not able to find any statistics on joint distributions.

Brute-force Probabilities Scheme 1

We investigate the success probability of an adversary, when trying to guess identifying attributes by brute-force, i. e., searching the whole value space. We assume the adversary will try most likely values (those registered by most users according to the value distribution in the population) first. Figure C.2 shows the number of combinations to test in order to cover a certain percentage of the user population. This corresponds to the costs of an adversary, as in Scheme 1, to try one combination, one KDF operation plus one DHT `get` operation are necessary. For single attributes between 180 and 3000 combinations are enough to find a target with 50% success probability (4600 to 60 Million combinations for 100%). When the combination of two attributes has to be guessed, this increases to around 10^7 combinations for 50% success probability and up to 10^{15} combinations to search the whole value space.

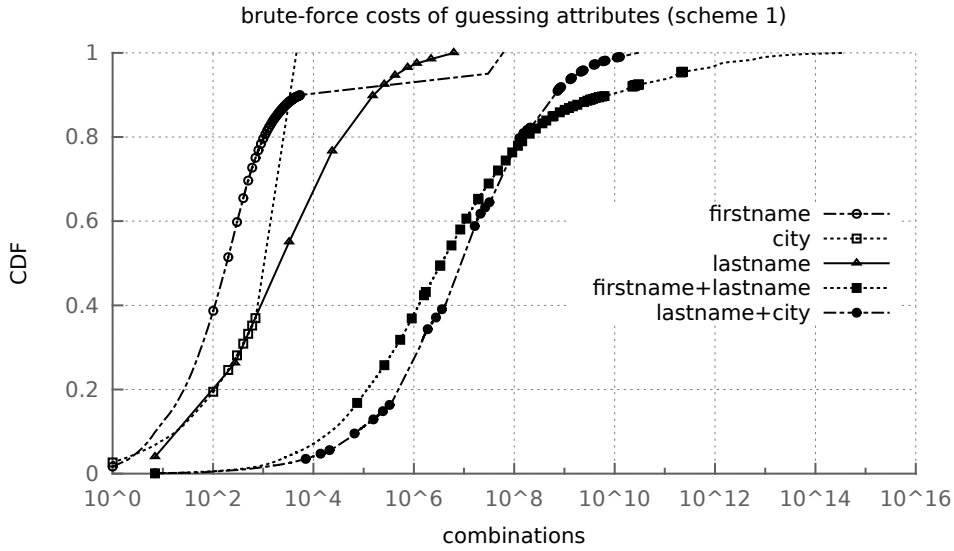


Figure C.2: CDF of brute-force success after trying a certain number of combinations (most likely ones first) for different attributes.

Brute-force Probabilities Scheme 2

In Scheme 2, bruteforcing works slightly differently. We assume, that the adversary knows some attributes (the fixed attributes that specify the subset to crawl, e. g., "workplace") and tries to guess other attributes (the identifying attributes). For each known attribute the adversary can issue a query and gets back a set of shares, each share having one bloom filter attached. Each share stems from a user who registered this specific attribute, i. e., a label-value combination (e. g., "workplace":"KTH") – several shares occur if several users registered the same combination (e. g., one from each user that registered their workplace as "KTH"). For an unknown attribute, the adversary will enumerate all possible values of the label-value combination (e. g., all possible lastname values for the attribute "lastname") and issue one DHT query each (after having performed a KDF operation to compute the DHT-key). This will result in one set of shares for each of the queries, again each share having one bloom filter attached. To know which share in each set should be picked, the bloom filters can be used to reduce the possible combinations. To test one combination, a second KDF operation (that might be tuned differently) has to be performed.

Figure C.3 shows the work to be done for a legitimate user searching for three attributes and an adversary, who knows one attribute and tries to guess two unknown attributes. Additionally, the ratio of the legitimate user's cost to the adversary's

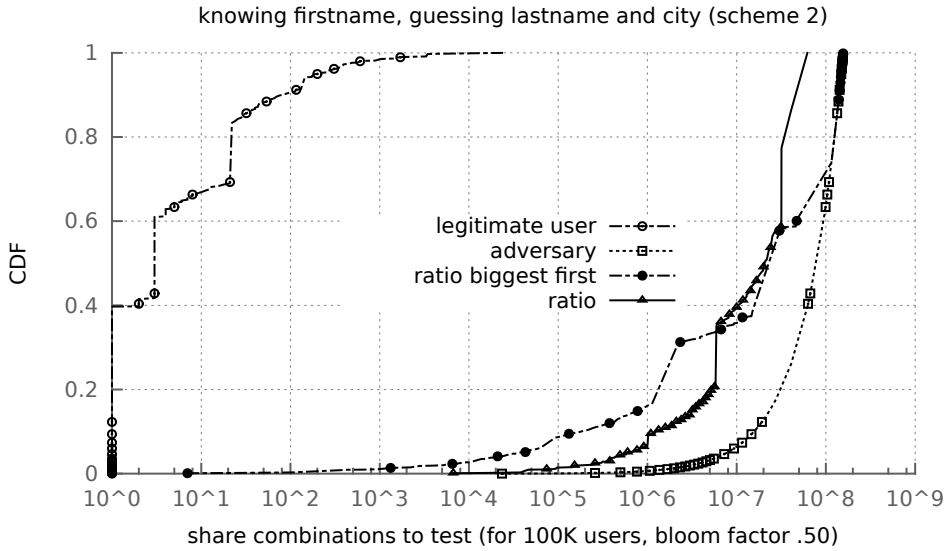


Figure C.3: Legitimate user searching for 3 attributes vs. adversary guessing 2 of them. Ratios depending on the adversaries strategy to search the value space.

cost is plotted, distinguishing two strategies of the adversary to search the value space: Either less popular values are tested first ("ratio") or more popular values first ("ratio biggest first").

Other Attacks

Existence Testing i. e., finding out if a user is registered in the system or not (without knowing enough attributes) is not possible in Scheme 1 and actively prevented in Scheme 2: Encrypting the user identifier under different keys (due to different salts) yields different ciphertexts and the bloom filters are salted differently. This is important as otherwise, searching e. g., for a certain firstname-lastname combination and getting the same ciphertext on the right hand side or similar bloom filters, reveals that there is a person with that firstname-lastname combination registered in the system, even if the person specified that more than two attributes are necessary to find her.

Search Query Data can give away information about the searcher (e. g., whom she is interested in) as well as previously unknown information about the searchee. A worst case example for the latter would be search queries that contain more information about the searchee than the searchee herself registered in the system.

An adversary observing these queries can at least probabilistically learn more information about the searchee.

This attack does, however, require the adversary to reverse the KDF operation that transformed the plaintext attribute combination (denoted *key* in the pseudocode) into a derived *dhtkey*. For Scheme 1, the search protocol tries longer combinations first, which are harder to reverse. For successful search operations, this prevents the searcher from issuing queries with a lower number of attributes than specified by the searchee as threshold. Unsuccessful search operations will, however, issue eventually queries with only one attribute in the *key*. For Scheme 2, every DHT-query is derived from only one attribute, so successfully reversing the KDF might be more likely in this case. One mitigation would be to obfuscate the query origin (e. g., by using a different Tor circuit for each query), but time-correlation attacks could still be successful.

Replaying an observed search query does not help an adversary if she is not able to reverse the KDF operation (transforming a *dhtkey* value back in a *key* value), because without the *key* value, she cannot decrypt the result of the search query.

Impersonation is not prevented by our protocols as they do not try to solve the general authentication problem. Although the *uid* should be signed by the searchee and its signature validated by the searcher after it was found (not described by our protocols), this does not keep an adversary from setting up a fake profile for John Doe and register the attributes "firstname:john", "lastname:doe" into the DHT, mapping it to the *uid* of the fake profile.

Discussion

The results presented in the previous section describe the gap between the search effort of a legitimate user and the cost of an adversary trying to find user identifiers despite knowing fewer attributes than required. For Scheme 1, the former is constant in terms of DHT operations, the latter depends on the number and kind of unknown attributes, as shown in Figure C.2. The adversary's costs for only one attribute are rather low, as expected. They can be tuned by KDF parameters but this will also affect the performance for legitimate users. The gap increases, however, combinatorially with the number of attributes the adversary has to guess. Already for two unknown attributes this might frustrate an attack: When tuning the KDF operations to take one second (delay for a legitimate user), an adversary with the same computational power as the user would need about 6 weeks to find the correct combination with 50% probability. The gap is not a global system parameter but can be tuned by each user individually (by choosing an individual threshold t for the registered information) but also depends on the adversary's knowledge about a target user. Scheme 2 can be tuned to achieve adversary costs comparable to that of Scheme 1, at the cost of slightly more work for legitimate users.

Apart from that, Scheme 1 has several advantages, compared to Scheme 2. It does not leak partial negative results, while Scheme 2, independently of any user thresholds, can reveal that a certain attribute combination is not registered in the system. For example, when searching for a certain lastname and workplace, and none of the shares of the two result sets are compatible according to the bloom filters, one learns that no user with this lastname registered this workplace. Furthermore, in Scheme 1 the adversary cannot make use of knowledge about other attributes of the user to decrease the search space for the identifying attributes. In Scheme 2, each additional attribute the adversary knows about the user, provides additional bloom filters to reduce the size of the result sets for the identifying attributes. Moreover, in Scheme 1 the user can specify even more fine-grained restrictions than only a minimum number of attributes. This makes weighting of attributes straightforward (see Section C.2), but can even be used to explicitly exclude certain attribute combinations that the user does not want to be found by.

The advantage of Scheme 2 is the lower number of items to store in the DHT for each user. In Scheme 1, besides the higher storage load for the DHT, this is mainly a problem for registering a user, as for each of the attribute combinations also one KDF has to be computed. While this could be solved by accepting a longer delay for the registration operation and let it run in the background, the higher number of combinations might, however, also incur problems for search queries in certain cases. When over-specifying the search target in Scheme 1 (i. e., providing a number of attributes that is greater than the searchee's threshold t), successively all subsets of the attributes have to be queried while in Scheme 2 the number of DHT queries is always equal to the number of specified search attributes.

Conclusion and Future Work

We presented two approaches to realize a targeted user search in a DOSN. The search protocols implement a knowledge threshold, allowing the users to protect their user identifier from adversaries that do not possess enough information about them while legitimate users, who know enough about the searchee, are able to find her. We described the protocols in detail, sketched a threat model, and evaluated selected properties using real world data. The evaluation yielded insights into the brute-force costs of an adversary, which depend on the user defined knowledge threshold and the knowledge of the adversary about the target user. The results suggest that for a subset crawling attack, the proposed protocols offer promising protection against an adversary that tries to brute-force at least two or three identifying attributes.

One open problem to be investigated in future work is the possibility of combining the two presented approaches. Building on Scheme 2, several attributes that have a rather small value space could be combined in the way it is done in Scheme 1, thus avoiding the high number of combinations while still leveraging the advantages of Scheme 1.

Acknowledgements

Oleksandr Bodriagov and Guillermo Rodríguez Cano contributed to joint discussions of the ideas in Section C.2. Some of the ideas were also discussed with Thomas Paul.

Article D

Event Invitations in Privacy-Preserving Decentralized Online Social Networks

Guillermo Rodríguez-Cano, Benjamin Greschbach, Sonja Buchegger

KTH Royal Institute of Technology
School of Computer Science and Communication
Stockholm, Sweden
{gurc, bgre, buc}@csc.kth.se

D

Abstract

Online social networks have an infamous history of privacy and security issues. One approach to avoid the massive collection of sensitive data of all users at a central point is a decentralized architecture.

An event invitation feature – allowing a user to create an event and invite other users who then can confirm their attendance – is part of the standard functionality of online social networks. We formalize security and privacy properties of such a feature like allowing different types of information related to the event (e. g., how many people are invited/attending, who is invited/attending) to be shared with different groups of users (e. g., only invited/attending users).

Implementing this feature in a privacy-preserving Decentralized Online Social Networks (DOSN) is non-trivial because there is no fully trusted broker to guarantee fairness to all parties involved. We propose a secure decentralized protocol for implementing this feature, using tools such as storage location indirection, ciphertext inferences and a disclose-secret-if-committed mechanism, derived from standard cryptographic primitives.

The results can be applied in the context of privacy-preserving DOSNs, but might also be useful in other domains that need mechanisms for cooperation and coordination, e. g., collaborative working environment and the corre-

sponding collaborative-specific tools, i. e., groupware, or computer-supported collaborative learning.

Introduction

The most common form of Online Social Networks (OSNs) are run in a logically centralized manner (although often physically distributed), where the provider operating the service acts as a communication channel between the individuals. Due to the popularity of these services, the extent of information the providers oversee is vast and covers a large portion of the population. Moreover, the collection of new types of sensitive information from each individual simply keeps increasing [122]. Users of these centralized services not only risk their own privacy but also the privacy of those they engage with. Whether intentional, or unintentional, data leakages [121], misuse [89] or censorship are some of the issues affecting the users.

Decentralization has been proposed to reduce the effect of these privacy threats by removing the central provider and its ability to collect and mine the data uploaded by the users as well as behavioral data. A Decentralized Online Social Network (DOSN) should provide the same features as those offered in centralized OSNs and at the same time it must preserve the privacy of the user in this different scenario. The latter is not straightforward, as in addition to the decentralization challenge itself, new privacy threats arise when the gatekeeper functionality of the provider that protects users from each other disappears [62].

One of the standard features of OSNs is the handling of event invitations and participation, i. e., a call for an assembly of individuals in the social graph for a particular purpose, e. g., a birthday celebration, demonstration, or meeting. There is usually metadata related to each event, such as date, location and a description. An implementation of this feature must provide security properties to the participants, e. g., that a user can verify that an invitation she received was actually sent by the organizer. Furthermore, it must support certain privacy settings. For example, an organizer could choose that only invited users learn how many other users were invited and that only after a user has committed to attend the event, she learns the identities of these other invited users.

Realizing this in a decentralized scenario is non-trivial because there is no Trusted Third Party (TTP) which all involved users can rely on. This is a problem, especially for privacy properties where information shall only be disclosed to users with a certain status, because any user should be able to verify the results to detect any possible cheating. In the example above, a neutral, trusted broker could keep the secret information (the identities of invited users) and disclose it only to users who committed to attend the event. This would guarantee fairness to both the organizer and the invited users. It becomes more challenging to implement this without a central TTP and still allowing different types of information about the event to be shared with different groups of users in a secure way.

Our contribution

We describe and formally define two basic and five more complex security and privacy properties for the event invitations feature.

We propose and discuss a distributed and privacy-preserving implementation of the event invitations feature without using a TTP. The suggested protocols cover all of our defined properties, considering 20 different parameter combinations for the tunable privacy properties.

We also describe three privacy-enhancing tools that we use in our implementation: storage location indirection, controlled ciphertext inference and a commit-disclose protocol. They are based on standard cryptographic techniques such as public key encryption, digital signatures and cryptographic hashes, and can be useful for other applications as well.

Paper Outline

We discuss related work in Section D.2, describe the problem of implementing the event invitation feature in a decentralized way and formalize security and privacy properties in Section D.3. Our proposed implementation together with privacy-enhancing tools follow in Section D.4, and we discuss this solution in Section D.5. We conclude with a summary and future work in Section D.6.

Related work

Groupware tools have been widely researched since they were first defined in 1978 by Peter and Trudy Johnson-Lenz [77]. Choosing between centralized and distributed implementations has been a major concern for these applications as pointed out in [109]. While the traditional model uses the client-server architecture [128, 87], there have been some projects on decentralized collaborative environments: Peer-to-pEer COLlaborative Environment [47], a P2P multicast overlay for multimedia collaboration in real-time, although synchronous; YCab [28], a mobile collaborative system designed for wireless ad-hoc networks; or a hybrid P2P architecture with centralized personal and group media tools in [146].

Security features in collaborative applications were already introduced in the popular client-server platform for businesses, IBM Notes/Domino (formerly Lotus Notes/Domino), to allow for usable authentication, and digital signature and encryption by means of a Public Key Infrastructure (PKI) to end-users [148]. Control policies in computer-supported collaborative work are considered in [112], including distributed architectures.

Protocol design guidelines in collaboration scenarios, where the privacy of a group member does not lessen by participating in the environment, have been studied and proposed in [82]. These guidelines aim at minimizing the amount of information a member has to provide to the group for the common activities, and making the protocols and the tasks transparent to everyone in the group.

Another type of related work lies within the domain of DOSNs [14, 39, 49]. To the best of our knowledge the event invitations feature has not been investigated in a privacy-preserving manner in this decentralized scenario.

Decentralizing The Event Invitation Feature

We already described the intuition of an event, where a group of people gathers with the intention of carrying out some activity. Now we more formally model the event invitation feature and desirable security and privacy properties. We denote the set of users as $U = u_1, \ldots, u_n$. The event invitation happens in three main stages:

- **Creation:** When a user $u_i \in U$ decides to create a new event e_k, she becomes the organizer o_{e_k} and creates the event object $event_k$ including different information, e. g., a description, date, time and location.
- **Invitation:** The organizer o_{e_k} selects the set of users to be invited to the event e_k, denoted by I_{e_k}, crafts the invitation objects $i_{e_k}^{u_j}$ for each of these invitees, and sends them to the respective users.
- **Commitment:** The invitees I_{e_k} have the chance of confirming the invitation, i. e., "commit" to attend the event e_k, by issuing commitment objects $c_{e_k}^{u_j}$. We denote the set of all attendees, i. e., the users who committed to the event e_k, as C_{e_k}.

Figure D.1 shows an example with eight users, $u_1 \ldots u_8$, where one of them, u_1, is the responsible organizer o_{e_k} of the event e_k. The organizer issues invitations to $u_2 \ldots u_6$, depicted with a dashed line. These users form the group of invitees, denoted with I_{e_k}. Invited users who confirm their attendance, (u_2, u_4 and u_6 in this example), provide a commitment to the organizer, depicted with a continuous line. They form the group of attendees, denoted with C_{e_k}.

A possible privacy setting could specify that invited users learn how many other users are invited but only attending users learn their identities. That is, u_3 and u_5 would learn that five users are invited (while this is kept secret from u_7 and u_8). u_2, u_4 and u_6 would additionally learn the identities of $I_{e_k} = u_2 \ldots u_6$.

System Model and Assumptions

In the following, we assume basic functionalities of popular OSNs to be available in a decentralized manner, such as user search [63] and user messaging [114]. We also assume that users are identified by a public key and the ability to verify the identity of other users via some sort of PKI, which can be realized in a decentralized manner, e. g., a "Web of Trust" model or a Bitcoin block-chain binding friendly usernames to public keys [57]. Moreover, we rely on a distributed storage featuring access right management, e. g., that a certain storage object is only writeable by a specific user,

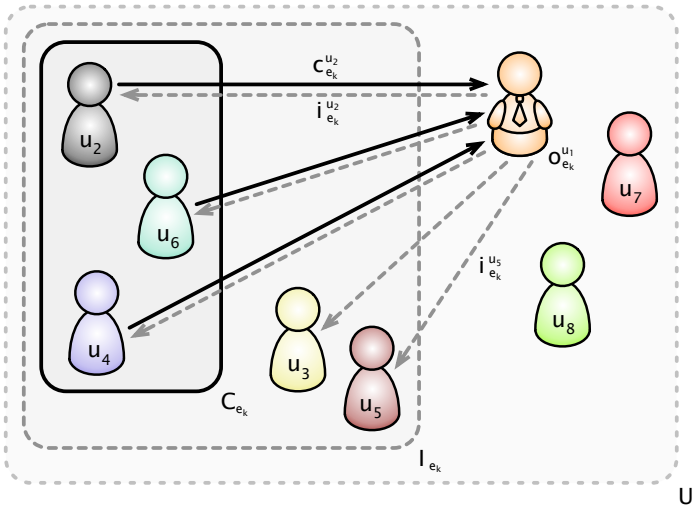


Figure D.1: Example of one event invitation.

and "append-only" storage objects, where new data can be appended, but existing data cannot be modified or removed without notice. The latter can be realized in a decentralized fashion, e. g., in a similar manner as the Bitcoin block-chain is secured against modifications [95].

Threat Model

We assume that users in all roles, e. g., invited users or the organizer of an event, might act maliciously, i. e., become adversaries. The capabilities of an adversary range from passively learning information accessible in that role (e. g., an invited user might have access to a list of all other invited users, depending on the privacy settings for the event), to actively interacting with other parties, e. g., writing arbitrary data to accessible storage objects or sending arbitrary messages to other users. We also assume that powerful adversaries might have the possibility to pervasively monitor a large fraction of the network traffic. While we try to mitigate threats like traffic analysis and correlation attacks arising from this, we cannot completely protect against them and come back to this in the discussion section. We do not assume that adversaries can subvert the storage layer. So we assume the availability of a secure distributed storage including features like append-only lists and authorization mechanisms, as mentioned above.

We want to keep malicious users from undermining the reliability of the event invitation feature for legitimate users. This means that an adversary should not be able to violate the security and privacy properties that we define in the next section. This comprises guaranteeing the authenticity and non-repudiation of state-

ments made by the involved parties, such as issued invitations or commitments. Furthermore it includes keeping information such as the identities of invited/attending users, the number of invited/attending users or a private event description secret from unauthorized users while guaranteeing its availability and authenticity for legitimate users. An example for the latter would be to keep an organizer from withholding or lying about the number of attending users. We do not focus on denial-of-service attacks and leave them for future work.

Security and Privacy Properties

A protocol for event invitations can comply with different security and privacy properties. We first list the following basic security properties:

- *A user u_j can prove that she was invited to the event e_k if and only if the organizer o_{e_k} invited u_j, i. e., issued an invitation $i_{e_k}^{u_j}$.*

  This property is two-sided and guarantees that a user cannot forge an invitation she did not get, while an organizer cannot deny that she invited a user. This implies that an invitation $i_{e_k}^{u_j}$ is tied to a user u_j that was chosen by the organizer o_{e_k} and cannot be transferred to another user.

- *An organizer o_{e_k} can prove that the invited user u_j committed to attend the event e_k if and only if u_j actually committed, i. e., issued a commitment $c_{e_k}^{u_j}$.*

  This property also has two sides. The organizer cannot forge a commitment of a user that did not commit to the event. And a user cannot deny that she committed to an event once she did so.

More challenging properties are those defining which groups of users are allowed to see what information, namely,

Invitee Identity Privacy (IIP)

For an event e_k, only a chosen set of users (e. g., U, I_{e_k}, C_{e_k} or only o_{e_k}) learns who else is invited (i. e., sees all members of I_{e_k}).

This property defines who can see information about who is invited to an event. This can be all users (U) or be restricted so that only other invited users see who else is invited (I_{e_k}). Another possibility is that even an invited user first learns who else is invited when she committed to attend (C_{e_k}). Finally, this information could be kept completely secret, so only the organizer o_{e_k} knows the complete list of invited users.

Invitee Count Privacy (ICP)

For an event e_k, only a chosen set of users (e. g., U, I_{e_k}, C_{e_k} or only o_{e_k}) learns how many users are invited (i. e., learns $|I_{e_k}|$).

This property is a variant of property IIP where the number of the invited people I_{e_k} is disclosed to a set of users (while the identities of the invited people might remain hidden).

Property IIP and ICP are closely related in the sense that if IIP holds for a certain set of users, then ICP trivially holds for the same set (and all its subsets – note the subset relation of the possible sets to choose from, $U \supseteq I_{e_k} \supseteq C_{e_k}$).

This constrains the possible combinations of these two properties' parameters. If, for example, for a certain event all invited users I_{e_k} should see who else was invited, i. e., property IIP with parameter choice I_{e_k}, then it does not make sense to choose that only the attendees C_{e_k} should learn the number of invited people, i. e., property ICP with parameter choice C_{e_k}, because the invited users can already derive this information from what they learn from property IIP.

Attendee Identity Privacy (AIP)

For an event e_k, only a chosen set of users (e. g., U, I_{e_k}, C_{e_k} or only o_{e_k}) learns who is attending (i. e., sees all members of C_{e_k}).

Attendee Count Privacy (ACP)

For an event e_k, only a chosen set of users (e. g., U, I_{e_k}, C_{e_k} or only o_{e_k}) learns how many users are attending (i. e., learns $|C_{e_k}|$).

Similarly to properties IIP and ICP, these two properties specify who can see information about the users who committed to attend an event. Property AIP defines who can see the identities of the attendees while property ACP defines to whom the number of attendees is disclosed. The same relation, regarding the possible parameter choices, as described for properties IIP and ICP, also holds here.

Attendee-only Information Reliability (AIR)

An invited user u_j can only get access to the private description $d^S_{e_k}$ of the event e_k once committed and the organizer o_{e_k} can only claim the attendance of the user u_j once the private description $d^S_{e_k}$ is available to u_j.

This property has two sides. First, a user u_j can only get access to information

> exclusive to the attendees C_{e_k}, i. e., the private description $d^S_{e_k}$ from the organizer o_{e_k} for an event e_k, if she has committed to attend. Second, and conversely, the organizer o_{e_k} can only claim that user u_j has committed to attend if she has made it possible for u_j to access the private description $d^S_{e_k}$.

Implementation

We now propose an implementation of the event invitation feature described in Section D.3 in a privacy-preserving DOSN. We assume that user identifiers u_i are public keys, and we will denote their corresponding private keys as u_i^S (where S stands for "secret").

System Components

The main components of the system are event objects, invitation objects and commitment objects as depicted in Figure D.2.

- **Event object:** When a user wants to create a new event, she first generates a public/private keypair e_k/e_k^S. The public key will become the identifier for the event and the user will be denoted as organizer o_{e_k}. She then assembles the event object $event_k$: She writes a public event description d_{e_k} and a private description $d^S_{e_k}$ that will be encrypted with a symmetric key PDK. She creates one list to store the invitation objects (*invite-list*) encrypted with a symmetric key ILK, another list for the commitment objects (*commit-list*) and one for disclosing secret information to committed users (*disclose-list*). The event object contains links ILL, CLL and DLL, pointing to the storage locations of these three lists. Additionally the organizer creates a list of public/private keypairs $rk_1/rk_1^S, \ldots, rk_n/rk_n^S$, to encrypt the entries on the commit-list, and includes the public keys in the event object. Moreover, the event object contains information about the chosen privacy settings.

  The organizer signs the public key of the event with her own user key to confirm that she is the organizer and signs the whole event object $event_k$ with the event's private key e_k^S. Therefore, an event object is composed as follows:

$$event_k = Sign_{e_k^S}(Sign_{u_i^S}(e_k)||u_i||d_{e_k}||Enc_{PDK}(d^S_{e_k}) \\ ||ILL||ILK||CLL||DLL||rk_1, \ldots, rk_n||\text{privacy settings})$$

  Some of the elements of the event object might, however, be encrypted with additional keys or only be hashes (made with a cryptographic hash function H, e. g., SHA-2 [61]) of the actual values. This depends on the chosen privacy settings and will be explained in more detail later.

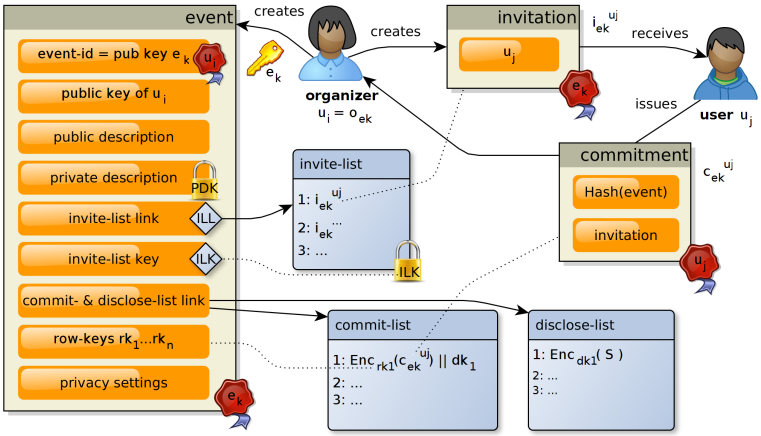


Figure D.2: Overview of the actors, system components and their relations.

- **Invitation object:** An invitation object is composed of the invitee's identifier u_j (her public key), signed by the organizer o_{e_k} with the event's private key e_k^S:

$$i_{e_k}^{u_j} = Sign_{e_k^S}(u_j)$$

- **Commitment object:** A commitment object is composed of the invitation object $i_{e_k}^{u_j}$ and the cryptographic hash of the event object $event_k$, both signed by the attending user u_j with her private key u_j^S as follows,

$$c_{e_k}^{u_j} = Sign_{u_j^S}(H(event_k)||i_{e_k}^{u_j})$$

Privacy Enhancing Tools

Before describing the implementation, we introduce tools that we will use several times.

Storage Location Indirection and Controlled Ciphertext Inference

If we want to make the size of a list, i. e., the number of its elements, available to a subset of users, but not the content of the list elements (in our scenario because each element contains a user identifiers), we can use storage location indirection and ciphertext inference: The list will not be stored together with the event object, but at a secret location in the distributed storage such that it can only be reached if the link to it is known. Additionally, the elements of the list will be encrypted so that the stored content can only be accessed if the encryption key is known.

This provides the possibility of a controlled information disclosure depending on the knowledge of a user: Users who do not know the link, learn nothing, neither the size nor the content of the list. Making the link to the list but not the encryption key available to a subset of users, enables these users to learn the size of the list (assuming a constant ciphertext size for each entry), while it does not give them any details about the contents stored. Users that received both the link and the encryption key, learn the content and can act as verifiers, checking that there are no invalid entries that incorrectly increase the perceived number of elements as seen by those users holding only the link but not the key.

Commit-Disclose Protocol

The organizer may want to share some information only with users who have committed to attend the event (attendees). To ensure fairness, the invited users need some guarantee that they can expect to receive the promised information when they commit to attend.

While this is easy to solve if both parties, the organizer and the invited users, trust a neutral third party that can act as broker, it becomes more difficult in our setting where we do not assume the existence of any TTP. So we base our solution on a significantly weaker trust assumption: the availability of append-only storage objects as described in Section D.3.

The aim of the protocol is to provide an invitee u_j who commits to the event e_k with a secret S held by the organizer o_{e_k}. It is composed of three main components, provided by the organizer of the event:

- **Commit-List**, a public and append-only storage object where invited users store their (encrypted) commitments.
- **Disclose-List**, a public readable, but only writeable by the organizer, append-only storage object where the organizer discloses (encrypted) secrets for the committed users.
- **Anchor Point**, a storage object (in our case the event object) serving as common entry point, referencing the commit-list and the disclose-list either directly by providing their storage locations, i. e., a commit-list link CLL and a disclose-list link DLL or indirectly by holding salted hashes of these storage locations (where DLL and CLL together with the salts are shared with a subset of users in another way). Additionally, a list of public keys $rk_1, \ldots, rk_n$, called row-keys, used to encrypt the entries on the commit-list are also stored here. All this information is signed by the organizer.

Each key in the row-keys list is intended for encrypting one entry of the commit-list. The corresponding private keys $rk_1^S, \ldots, rk_n^S$, are held by the organizer. The protocol runs in three phases:

- **Commit Phase:** If the user u_j wants to commit to attend the event e_k, she looks up the commit-list and finds the next free row – let this have index l. She then looks up the corresponding row key rk_l in the event object.

  Finally, she crafts a commitment $c_{e_k}^{u_j}$, creates a fresh keypair dk_l^P/dk_l^S (disclose key, later used by the organizer to encrypt the secret information) and writes the following entry to row l of the commit-list: $Enc_{rk_l}(c_{e_k}^{u_j})||dk_l^P$ that is the commitment, encrypted with the row-key, together with the public disclose key in plain.

- **Disclose Phase:** When the organizer o_{e_k} sees that a new row l has been added to the commit-list, she tries to decrypt the first entry, using the secret row key rk_l^S. If this succeeds and the commitment is valid the organizer writes the secret information, encrypted with the provided disclose key to row l of the disclose-list, i. e., $Enc_{dk_l^P}(S)$. If the decryption fails or the commitment is invalid, the organizer publishes the secret row-key of row l in the disclose-list instead, i. e., rk_l^S, thus proving to everybody who can access the lists that she was not obliged to disclose the secret information to the creator of row l.

- **Blame Phase:** If the organizer misbehaves and does not provide a protocol-abiding user with the secret information after a reasonable amount of time, the user can blame the organizer. She does this by publishing a blame-entry in the commit-list, referring to the row l and disclosing the secret disclosure key dk_l^S. Thus everybody who can access the lists can see that she did not receive the secret information encrypted to the disclosure key she provided in row l. It can be assumed that the commitment (which cannot be decrypted by the verifying public) was correct, as otherwise the organizer would have published the secret row-key of row l.

In this way, the commit-disclose protocol does not keep the organizer from cheating, but it allows the user to reliably blame the organizer if it is the case.

Basic Security Properties

The basic security properties are fulfilled by the construction of an event, invitations and commitments described in Section D.4 and the guarantees of the PKI. The first basic security property is fulfilled because an invitation $i_{e_k}^{u_j}$ for a user u_j is created by using the event's private key e_k^S, owned by the organizer o_{e_k} to sign the invited user's identifier. The invitee cannot forge the event's key and the organizer cannot deny having issued the invitation because the signature used to sign the invitation is publicly verifiable. The second basic security property is also fulfilled because an organizer o_{e_k} cannot forge a commitment $c_{e_k}^{u_j}$ as she is not able to forge another users' signature. A user u_j, having sent the commitment $c_{e_k}^{u_j}$ to the organizer

o_{e_k}, cannot deny the commitment as her signature is again publicly verifiable and binding to the event e_k.

Invitee Identity Privacy and Invitee Count Privacy (ICP)

In order to implement properties IIP and ICP, we let the organizer o_{e_k} store all invitation objects for the event in the invite-list. Retrieving the list requires knowledge of the invite-list link ILL, and in order to decrypt it, the symmetric invite-list key ILK must be known beforehand.

Knowledge of the link ILL is equivalent to learning the total number of invitations, even if the decryption key ILK is unknown because the number of invitations can be inferred from the size of the ciphertext in the list. Knowledge of the encryption key ILK allows learning the identities of the invited users I_{e_k} because the invitations $i_{e_k}^{u_j}$ store the user identifiers in plain text.

If the organizer o_{e_k} wants to make the identifiers of the invitees I_{e_k}, or the amount of them, i. e., $|I_{e_k}|$, available to all users U, she will publish ILL or ILK in plain text together with the event object $event_k$. Making this information available only for invitees I_{e_k} can be realized by the organizer privately sharing it with the invited users. In order to share the decryption key ILK only with the committed users C_{e_k}, the commit-disclose protocol can be used, while the link ILL is then either available publicly (i. e., choosing U for property ICP), shared only with the invitees (i. e., choosing I_{e_k} for ICP) or kept secret and only shared with the committed users together with ILK (i. e., choosing C_{e_k} for ICP).

It is also possible to avoid sharing any information about the invitations by keeping ILL and ILK secret, i. e., choosing o_{e_k} both for properties IIP and ICP. When the identities should not be known to anyone but the number of invitees should be made public to a subset of users (i. e., choosing o_{e_k} for property IIP), the link ILL will be shared with the respective users and a particular encryption scheme for the invite-list is employed: Instead of encrypting the invite-list as a whole, we encrypt its individual entries with the public keys of the recipient of the invitation stored at each entry. Thus, the invited users can verify that their own invitation is included in the list. However, this only allows for a weak verification of the correctness of the list, i. e., it provides an upper-bound of the size of the list, because the organizer o_{e_k} can add invalid or dummy entries (e. g., to artificially increase the perceived number of invitees to the event).

A summary of how ILL and ILK are shared depending on the choice of parameters for properties IIP and ICP is shown in Table D.1. Note that the row describing the privacy settings IIP: C_{e_k}, ICP: I_{e_k}corresponds to the example mentioned in the introduction and Section D.3.

Table D.1: Sharing of ILL and ILK as per the IIP and ICP settings. P = publicly available in $event_k$, I = privately shared with I_{e_k}, C = shared only with C_{e_k} (via the commit-disclose protocol), S = fully secret (only o_{e_k} knows about it) and S* = special encryption scheme for the invite-list.

Settings		**Implementation**	
IIP	ICP	ILL	ILK
U	U	P	P
I_{e_k}	U	P	I
	I_{e_k}	I	I
C_{e_k}	U	P	C
	I_{e_k}	I	C
	C_{e_k}	C	C
o_{e_k}	U	P	S*
	I_{e_k}	I	S*
	C_{e_k}	C	S*
	o_{e_k}	S	S

Attendee Identity Privacy (AIP) and Attendee Count Privacy (ACP)

To implement the AIP and ACP properties, we mainly use the commit-disclose protocol. The link to the commit-list CLL can be shared publicly in the event object $event_k$ except for those cases where the count of attendees $|C_{e_k}|$ must be kept private. In this situation, if the invitees I_{e_k} are allowed to learn $|C_{e_k}|$, CLL is shared privately with them. Alternatively, the organizer can add dummy entries in the list to hinder inferences from the number of (encrypted) entries. When not even attendees should learn how many other users are attending, dummy entries in the commit-list are the only solution as the CLL must always be shared with all invitees, so that they can commit if they want to attend.

Dummy entries follow the pattern of usual entries, i. e., random data with a specific size to fake an encrypted commitment object and a public key in the commit-list, and random data in the disclose-list to fake an encrypted secret. All users who hold the private row-keys can identify them because the first part of a dummy entry in the commit-list cannot be decrypted with the respective row-key, while those users without the private row-keys cannot distinguish dummy entries from real ones as the ciphertext structure looks the same for all of them.

When the link CLL should not be shared publicly in the event object $event_k$, a salted hash of the link will be stored instead so that the organizer o_{e_k} cannot cheat by sharing different links with different groups of users. As the event object is unique per event and group of invitees, the invited users can check they all got

Table D.2: Sharing of CLL and $rk_1^S \dots rk_n^S$ as per the AIP and ACP settings. P = publicly available in $event_k$, I = privately shared with I_{e_k}, C = shared only with C_{e_k} (via the commit-disclose protocol), S = fully secret (only o_{e_k} knows about it).

Settings		**Implementation**			
AIP	ACP	CLL	$rk_1^S ... rk_n^S$	dummies	notes
U	U	P	P	-	
I_{e_k}	U	P	I	-	
	I_{e_k}	P/I	I	if CLL public	
C_{e_k}	U	P	C	-	
	I_{e_k}	P/I	C	if CLL public	
	C_{e_k}	P	C	necessary	
o_{e_k}	U	P	S	-	
	I_{e_k}	I	S	-	
	C_{e_k}	-	-	-	not possible
	o_{e_k}	P	S	necessary	

the same link from the organizer by comparing it with the hash value in $event_k$.

Otherwise the implementation varies only in how the private row-keys are disclosed, as they protect the commitments in the commit-list: If all users U are allowed to learn who is attending, the private row-keys will be public, i. e., the rows do not need to be encrypted. If only the invited users I_{e_k} should see the identities of the attendees, the private row-keys will be shared with the invitees directly. And if only the attending users should learn about the identities of other attendees, the private row-keys are disclosed using the commit-disclose protocol.

This way we are able to implement all possible parameter combinations of the AIP and ACP properties, except for the combination AIP: o_{e_k}, ACP: C_{e_k}. For this case, i. e., AIP: o_{e_k}, nobody except the organizer should learn the identities of the committed users, so the private row-keys have to be kept secret. And as not even invitees (who need to know CLL to be able to commit to the event) should learn the count of attendees, the organizer would need to add dummy entries on the commit-list to hide the count of attendees from the invitees. But this will also hide it from the attendees, as they do not have the private row-keys to tell apart dummy entries from normal entries, so ACP: C_{e_k} is not fulfilled.

A summary of how CLL and the private row-keys $rk_1^S \dots rk_n^S$ are shared depending on the settings for properties AIP and ACP is shown in Table D.2.

Attendee-only Information reliability (AIR) Property

To implement this property, we will again use the commit-disclose protocol. The organizer o_{e_k} shares a private description $d_{e_k}^S$, encrypted with the key PDK, with

the committed users C_{e_k}. The key is shared with these users in the disclose-list as soon as they store a valid commitment $c_{e_k}^{u_j}$ in the commit-list. The organizer o_{e_k} cannot have different private descriptions for groups of attendees of the same event e_k because they will all see the same ciphertext in the event object $event_k$. A cheating organizer o_{e_k} will be caught in the same manner as described above: if a user u_j commits and receives an invalid decryption key PDK, she will publish the private disclose key dk_i^S to prove that she did not receive the promised private description $d_{e_k}^S$.

Discussion

The implementation presented realizes the event invitation feature in a decentralized system and fulfills the requirements of all of the defined security and privacy properties. Except for one parameter combination of the attendee identity/count privacy properties we were able to present implementation solutions for all possible choices of the tunable properties IIP, ICP, AIP and ACP.

An honest but curious user does not learn anything more than what is specified by the privacy settings.

A general limitation of our approach is, however, that for all properties based on the commit-disclose protocol, a malicious organizer is still able to cheat. But it disincentives her to do so as it provides a reliable cheating detection mechanism and offers the affected users the possibility to blame a cheating organizer – either publicly or in front of a chosen set of users, e. g., only other invitees of the event. We consider this an effective protection in the social scenarios that we see as possible application contexts of the event invitation feature. User identifiers are long-lived there and costly to change (as all friends have to be informed about a new identity), so we assume users care about their reputation and will try to avoid being exposed as misbehaving. Another limitation of our approach is the general problem of information usage control, i. e., insiders can always leak information to parties that should not learn this information according to a chosen privacy setting. For example, if only the invitees should learn the identities of other invited users, this can be violated by an invitee simply publishing the invite-list.

Some of the privacy protections are not secure against very powerful adversaries. For example the link obfuscation technique described in Section D.4 relies on the unlinkability of the encrypted list object and the event object. This will be decreased by access patterns of invited users (if they are known), the structure/size of the list object (if distinguishable from other storage objects) and the entropy of the addressing scheme for storage objects. An adversary with the capability to pervasively monitor a large fraction of network traffic might be able to correlate requests for a certain event object and related list objects.

Finally, depending on the choice of privacy settings, the protocols not only allow the participants, i. e., organizer, invitees and attendees, to verify each others' claims, but also, to show the proof to an outsider. Such a process can be implemented in

a client and used as one of the inputs for a reputation system, although this is out of the scope of this work.

Conclusion and Future Work

We have described and formalized a set of security and privacy properties for the event invitations feature in DOSNs, such as invitee/attendee identity privacy (who learns the identities of the invitees/attendees), invitee/attendee count privacy (who learns the count of invitees/attendees), and Attendee-only Information Reliability (availability of information exclusive to the attendees).

We described privacy enhancing tools, such as storage location indirection (to control not only who can decrypt an object but also who can see the ciphertext), controlled ciphertext inference (to allow a controlled information leak, e. g., about the size of an encrypted object to parties not able to decrypt the content) and a commit-disclose protocol to disclose a secret only to users who committed to attend an event and to detect a misbehaving party. Using these tools together with standard cryptographic primitives, we proposed a TTP-free architecture and decentralized protocols to implement the event invitation feature in a DOSN and analyzed the usability and privacy implications.

The results can be applied in the context of privacy-preserving DOSNs, but might also be useful in other domains such as collaborative working environment and their corresponding collaborative-specific tools, i. e., groupware, for example, to perform tasks on shared documents. Another relevant domain is Massive Open Online Courses (MOOC), for example, when restricting the access to lecture material of an online course to the registered students.

Possible future work includes evaluation of the performance, extending the security and privacy properties to include plausible deniability, anonymity or revocation, and extending the functionality of the feature to consider transferable invitation-rights or multiple organizers. Plausible deniability properties can be important when organizing political events. At the same time, it will probably introduce trade-offs with respect to the authenticity guarantees provided by the properties presented in this paper, e. g., the correctness of the attendee-count. Transferable invitation-rights would allow the organizer to specify a set of initially invited users, who then in turn can invite their friends to the event as well (but maybe limited to a certain number of hops in the social graph).

Acknowledgments

This research has been funded by the Swedish Foundation for Strategic Research grant SSF FFL09-0086 and the Swedish Research Council grant VR 2009-3793.

Article E

The Effect of DNS on Tor's Anonymity

Benjamin Greschbach*[1], Tobias Pulls*[2], Laura M. Roberts*[3], Philipp Winter*[3], Nick Feamster[3]
[1]KTH Royal Institute of Technology, [2]Karlstad University, [3]Princeton University
*All four authors contributed substantially, and share first authorship. The names are ordered alphabetically.

E

Abstract

Previous attacks that link the sender and receiver of traffic in the Tor network ("correlation attacks") have generally relied on analyzing traffic from TCP connections. The TCP connections of a typical client application, however, are often accompanied by DNS requests and responses. This additional traffic presents more opportunities for correlation attacks. This paper quantifies how DNS traffic can make Tor users more vulnerable to correlation attacks. We investigate how incorporating DNS traffic can make existing correlation attacks more powerful and how DNS lookups can leak information to third parties about anonymous communication. We (*i*) develop a method to identify the DNS resolvers of Tor exit relays; (*ii*) develop a new set of correlation attacks (DefecTor attacks) that incorporate DNS traffic to improve precision; (*iii*) analyze the Internet-scale effects of these new attacks on Tor users; and (*iv*) develop improved methods to evaluate correlation attacks. First, we find that there exist adversaries that can mount DefecTor attacks: for example, Google's DNS resolver observes almost 40% of all DNS requests exiting the Tor network. We also find that DNS requests often traverse ASes that the corresponding TCP connections do not transit, enabling additional ASes to gain information about Tor users' traffic. We then show that an adversary that can mount a DefecTor attack can often determine the website that a Tor user is visiting with perfect precision, particularly for less popular websites where the set of DNS names associated with that website may be unique to the site. We also use the Tor Path Simulator (TorPS) in combination with traceroute data from vantage points co-located with Tor

exit relays to estimate the power of AS-level adversaries that might mount DefecTor attacks in practice.

Introduction

We have yet to learn how to build anonymity networks that resist global adversaries, provide low latency, and scale well. Remailer systems such as Mixmaster [93] and Mixminion [41] eschew low latency in favor of strong anonymity. In contrast, Tor [45] trades off strong anonymity to achieve low latency; Tor therefore enables latency-sensitive applications such as web browsing but is vulnerable to adversaries that can observe traffic both entering and exiting its network, thus enabling deanonymization. Although Tor does not consider global adversaries in its threat model, adversaries that can observe traffic for extended periods of time in multiple network locations (*i*.e., "semi-global" adversaries) are a real concern [51, 76]; we need to better understand the nature to which these adversaries exist in operational networks and their ability to deanonymize users.

Past work has quantified the extent to which an adversary that observes TCP flows between clients and servers (*e*.g., HTTP requests, BitTorrent connections, and IRC sessions) can correlate traffic flows between the client and the entry to the anonymity network and between the exit of the anonymity network and its ultimate destination [76, 94]. The ability to correlate these two flows—a so-called *correlation attack*—can link the sender and receiver of a traffic flow, thus compromising the anonymity of both endpoints. Although TCP connections are an important part of communications, the Domain Name System (DNS) traffic is also quite revealing: for example, even loading a single webpage can generate hundreds of DNS requests to many different domains. No previous analysis of correlation attacks has studied how DNS traffic can exacerbate these attacks.

DNS traffic is highly relevant for correlation attacks because it often traverses completely different paths and autonomous systems (ASes) than the subsequent corresponding TCP connections. An attacker that can observe occasional DNS requests may still be able to link both ends of the communication, even if the attacker cannot observe TCP traffic between the exit of the anonymity network and the server. Figure E.1 illustrates how an adversary may monitor the connection between a user and the guard relay, and between the exit relay and its DNS resolvers or servers. This territory—to-date, completely unexplored—is the focus of this work.

We first explore how Tor exit relays resolve DNS names. By developing a new method to identify all exit relays' DNS resolvers, we learn that Google currently sees almost 40% of all DNS requests exiting the Tor network. Second, we investigate which organizations can observe DNS requests that originate from Tor exit relays. To answer this question, we emulate DNS resolution for the Alexa top 1,000 domains from several ASes. We find that DNS resolution for half of these domains traverses numerous ASes that are not traversed for the subsequent HTTP connection to

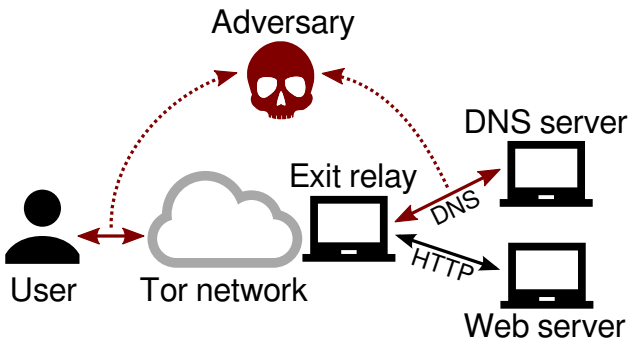


Figure E.1: Past traffic correlation studies have focused on linking the TCP stream entering the Tor network to the one(s) exiting the network. We show that an adversary can also link the associated DNS traffic, which can be exposed to many more ASes than the TCP stream.

the web site. Next, we show how the ability to observe DNS traffic from Tor exit relays can augment existing website fingerprinting attacks, yielding perfectly precise DefecTor[1] attacks for unpopular websites. We further introduce a new method to perform traceroutes from the networks where exit relays are located, making our results significantly more accurate and comprehensive than previous work. Finally, we use the Tor Path Simulator (TorPS) [75] to investigate the effects of Internet-scale DefecTor attacks.

We demonstrate that DNS requests significantly increase the opportunity for adversaries to perform correlation attacks. This finding should encourage future work on correlation attacks to consider both TCP traffic and the corresponding DNS traffic; future design decisions should also be cognizant of this threat. Our work (*i*) serves as guidance to Tor exit relay operators and Tor network developers, (*ii*) improves state-of-the-art measurement techniques for analysis of correlation attacks, and (*iii*) provides even stronger justification for introducing website fingerprinting defenses in Tor. To foster future work and facilitate the replication of our results, we publish both our code and datasets.[2] In summary, we make the following contributions:

- We show how existing website fingerprinting attacks can be augmented with observed DNS requests by an AS-level adversary to yield perfectly precise DefecTor attacks for unpopular websites.

- We develop a method to identify the DNS resolver of exit relays. We find that Tor exit relays comprising 40% of Tor's exit bandwidth rely on Google's public DNS servers to resolve DNS queries.

[1] The acronym is short for DNS-enhanced fingerprinting and egress correlation on Tor.
[2] Our project page is available at `https://nymity.ch/tor-dns/`.

- We quantify the extent to which DNS resolution exposes Tor users to additional AS-level adversaries who are not on the path between the sender and receiver. We find that for the Alexa top 1,000 most popular websites, many ASes that are on the paths between the exit relay and the DNS servers required to resolve the sites' domain names are not on the path between the exit relay and the website.
- We develop a new measurement method to evaluate the extent to which ASes are on-path between exit relays and DNS resolvers. We use the RIPE Atlas [111] platform to achieve previously unprecedented path coverage and accuracy for evaluating the capabilities of AS-level adversaries.

The rest of this paper is organized as follows. Section E.2 presents background, and Section E.3 relates our study to previous work. In Section E.4, we shed light on the landscape of DNS in Tor. Section E.5 discusses our DefecTor attacks, which we evaluate in Section E.6. We then model the Internet-scale effect of our attacks in Section E.7. Finally, we discuss our work in Section E.8 and conclude the paper in Section E.9.

Background

We now provide an introduction to the Tor network, website fingerprinting attacks, as well as how the Tor network implements DNS resolution.

The Tor network The Tor network is an overlay network that anonymizes TCP streams such as web traffic. As of August 2016, it comprises approximately 7,000 relays and about two million users. The hourly published network consensus summarizes all relays that are currently online. Clients send data over the Tor network by randomly selecting three relays—typically called the guard, middle, and exit relay—that then form a virtual tunnel called a *circuit*. The guard relay learns the client's IP address, but not its web activity, while the exit relay gets to learn the client's web activity, but not its IP address. Relays and clients talk to each other using the Tor protocol, which uses 512-byte *cells*. Finally, each relay is uniquely identified by its fingerprint—a hash over its public key.

Website fingerprinting attacks The Tor network encrypts relayed traffic as it travels from the client to the exit relay. Therefore, intermediate parties such as the user's Internet service provider (ISP) cannot read the contents of any packet. Tor does not, however, protect other statistics about the network traffic, such as packet inter-arrival timing, directions, and frequency. The ISP can analyze these properties to infer the destinations that a user is visiting. The literature calls this attack *website fingerprinting*.

Past work evaluated website fingerprinting attacks in two settings: a *closed-world* setting consists of a set of n monitored websites, and the attacker tries to

learn which among all n sites the user is visiting with the notable restriction that the user can only browse to one of the n websites. The *open-world* setting is more realistic: the user can browse to unmonitored sites in addition to the monitored sites. Unmonitored sites are, per definition, not known to the attacker; thus, the attacker's traffic classifier cannot train on unmonitored sites the user visits. The attacker's classifier can train on whatever unmonitored sites it chooses, as long as the classifier has not trained on an unmonitored site used for testing. Two relevant metrics in the open-world setting are *recall* and *precision*. Recall measures the probability that a visit to a monitored site will be detected, while precision measures the probability that a classification by the classifier of a visit to a monitored site (positive test outcome) is the correct one. Consider a classifier with 0.25 recall and 0.5 precision: on average, every fourth visit by the user to a monitored site will be detected, and half of the classifications by the classifier will be wrong. Errors can either classify a monitored site as unmonitored (lowering recall) or vice versa (lowering precision). Mistaking one monitored site for another is less likely [134].

Wa-kNN is a website fingerprinting attack by Wang *et* al. [135] that uses a k-nearest neighbor classifier with a custom weight-learning algorithm, WLLCC [134, § 3.2.5]. From packet traces between a Tor client and its guard, Wa-kNN extracts a number of features to classify each website. Useful features include the number of outgoing packets and bursts of packets in the same direction. In the training phase, WLLCC adjusts weights of features extracted from sites of known classes such that the distance between instances of the same site (class) are minimized (collapsed). In the testing phase, Wa-kNN determines the distance of a testing traffic trace to all known training traces. The distance calculation results in the k nearest classes: if all classes are the same, then the testing trace is classified as that class, otherwise it is classified as unmonitored. In the open-world setting, one class represents all unmonitored sites both during training and testing. By increasing k, Wa-kNN can trade decreased recall for increased precision. We set $k = 2$ when using Wa-kNN for higher recall since DefecTor is a highly precise attack.

Tor could eliminate website fingerprinting attacks with encrypted, constant-bitrate channels between a Tor client and its guard; other anonymity networks could use a similar technique. Unfortunately, the Tor network's limited spare capacity does not allow for such a throughput-intensive defense, but some research has worked on making this type of defense more efficient [29, 79, 134, 105].

How Tor resolves DNS requests Tor clients must send DNS requests over Tor to prevent DNS leakage (*i*.e., having a DNS request travel over an unencrypted channel as opposed to over Tor itself). Tor does not transport UDP traffic, but it implements a workaround to wrap DNS requests into Tor cells. Using the SOCKS protocol, applications can instruct the Tor client to establish a circuit to a given domain and port.[3] After the user types in a domain, say example.com, the Tor

[3]The SOCKS versions 4a and 5 support connection initiation using domain names in addition to IP addresses.

browser establishes a connection to the SOCKS proxy exposed by the local Tor client. The Tor client then selects an exit relay whose exit policy supports example.com and port 443. Next, the client sends a `RELAY_BEGIN` Tor cell to the exit relay, instructing it to first resolve example.com, and then establish a TCP connection to the resolved address at port 443 [44, § 6.2]. After successfully establishing a connection, the exit relay responds with a `RELAY_CONNECTED` cell. The client can then exchange data with its intended destination. Another type of cell, `RELAY_RESOLVE`, supports pure name resolution, without establishing a subsequent TCP connection [44, § 6.4]. The exit relay responds with a `RELAY_RESOLVED` cell.

Exit relays send their DNS requests to the system resolver, which Linux systems read from `/etc/resolv.conf`. Tor does not modify the system resolver and uses whatever the exit relay operator configured, such as the ISP's resolver, or public resolvers such as Google's public DNS resolver 8.8.8.8. As of August 2016, exit relays cache DNS responses to speed up repeated lookups. The caching layer for Tor clients, however, is off by default to prevent tracking attacks due to modified DNS responses [91].

Related Work

This paper combines traffic analysis methods for correlation attacks with website fingerprinting attacks; we discuss related work in each of these two areas.

Traffic analysis and correlation attacks

Tor's threat model excludes global adversaries [45], but the practical threat of such adversaries is an important question that the research community has spent considerable effort on answering. In 2004, when the Tor network comprised only 33 relays, Feamster and Dingledine investigated the practical threat that AS-level adversaries pose to anonymity networks [52]. The authors considered an attacker that controls an AS that is traversed both for ingress and egress traffic, allowing the attacker to correlate both streams. Using AS path prediction [59], Feamster and Dingledine found that powerful tier-1 ISPs reduce location diversity of anonymity networks. In 2007, Murdoch and Zieliński drew attention to IXP-level adversaries, a class of adversaries that was missing in Feamster and Dingledine's work [94]. Murdoch and Zieliński showed that IXP adversaries are able to correlate traffic streams, even in the presence of packet sampling rates as low as one in 2,000.

In 2013, Johnson *et* al. [76] presented the first large-scale study on the risk of Tor users facing relay-level and AS-level adversaries. The authors developed TorPS [75] that simulates Tor circuits for a number of user models. By combining TorPS with AS path prediction, Johnson *et* al. could answer questions such as the average time until a Tor user's circuit is deanonymized by an AS or IXP. Most recently in 2016, Nithyanand *et* al. [99] used AS path prediction to evaluate the practical threat faced by users in the top ten countries using Tor. In 2015, Juen *et* al. [80] examined the

accuracy of path prediction algorithms that prior work [76, 52] used to estimate the threat of correlation attacks. The authors compared AS path predictions to millions of traceroutes, initiated from 25% of Tor relays by bandwidth at the AS level, and found that only 20% of predicted paths matched the paths observed in traceroutes. Juen *et* al. could not consider the reverse path in traceroutes. In 2015, Sun *et* al. [123] addressed this shortcoming; although past work treated routing as static, Sun *et* al. showed that the dynamic nature of Internet routing makes AS-level adversaries stronger than previous work had considered.

We improve on previous work in two significant ways: (*i*) we are the first to evaluate how DNS traffic exacerbates traffic correlation attacks, both in concept and in practice; and (*ii*) we develop and deploy a scalable, sustainable version of the measurement method proposed by Juen *et* al. [80]. Our method uses the volunteer-run RIPE Atlas measurement platform [111], as opposed to relying on relay operators to run third-party scripts. This approach allows us to fully automate our method and achieve previously unprecedented scale.

Website fingerprinting

In 2009, Herrmann *et* al. [69] demonstrated the first website fingerprinting attack against anonymity systems—including Tor—in a closed-world setting. In 2011, Panchenko *et* al. [101] greatly improved on Herrmann *et* al.'s detection rate and provided insight into an open-world setting. In 2012, Cai *et* al. [31] improved on previous work by proposing an attack that used Hidden Markov Models to determine whether a sequence of page requests all come from the same site. The authors used an open-world setting for their evaluation. Wang and Goldberg [136] proposed an improved attack that employed a new method for data gathering. In 2014, Wang *et* al. [135] further improved on their results with a k-nearest neighbor classifier Wa-kNN and a custom weight-learning algorithm (WLLCC [134, § 3.2.5]) that in several rounds determine the optimal weights for features extracted from traffic traces. Cai *et* al. [30] determined which traffic features provide the most predictive power to detect websites, proved a lower bound of any defense that achieves a certain level of security, and provided a framework to investigate the performance of website fingerprinting attacks. Juarez *et* al. [78] showed that all previous attacks made several simplifying assumptions; the work suggested that attacks are still difficult to run outside a lab setting as an attacker will have to consider operating system differences, page changes, and background traffic. Recently, in 2016, Wang and Goldberg addressed many practical roadblocks to website fingerprinting, such as noisy data and maintaining a training set, further highlighting the need for website fingerprinting defenses in Tor [137].

Panchenko *et* al. [100] showed that web*page* fingerprinting (*i*.e., fingerprinting of any page on a site) is significantly harder than web*site* fingerprinting (*i*.e., fingerprinting of only the start page of a site). Hayes and Danezis proposed k-fingerprinting, an attack with notably better performance than Wa-kNN even in the face of defenses [67]. Their attack retains 30% accuracy in a closed-world set-

ting against the WTF-PAD defense by Juarez *et* al. [79]—a prime candidate for deployment in Tor [105]—at the cost of 50% bandwidth overhead. Juarez *et* al. used Wa-kNN to evaluate WTF-PAD and set $k = 5$, as recommended by Wang *et* al. for an optimal trade-off between recall and the false positive rate.

In our work, we show how to correlate and use observed DNS requests in concert with website fingerprinting attacks, which significantly improves precision for web*site* fingerprinting. In scenarios where precision is paramount, DefecTor attacks pose an even bigger threat than website fingerprinting attacks from attackers that can observe even a modest fraction of DNS traffic from the Tor network. Mitigating the two DefecTor attacks that we present has implications for the design of website fingerprinting defenses: open-world evaluations of the website fingerprinting defense should minimize recall even when the website fingerprinting attack is tuned to sacrifice precision for recall. In the case of Wa-kNN, this means a low k: our results are based on $k = 2$.

Understanding the Landscape

Before explaining our attack, we need to better understand how Tor performs DNS resolution. We begin by investigating how common it is for adversaries to be able to observe DNS requests but *not* subsequent TCP connections of Tor users (Section E.4). We then seek to understand how these results connect to the Tor network by determining the DNS resolvers used by exit relays (Section E.4).

Quantifying the additional AS exposure of DNS queries

Adversaries that can observe both DNS and subsequent TCP traffic (*e.g.*, the ISP of an exit relay) gain no benefit from seeing the client's DNS traffic, since TCP traffic is sufficient to mount correlation attacks [94]. In this work, we consider adversaries that can observe traffic entering the Tor network and *some* DNS requests exiting the network—such as requests addressed to DNS root servers—but *not* subsequent TCP traffic from exit relays. We first determine the prevalence of these adversaries by measuring the number of ASes that DNS queries traverse versus the number of ASes subsequent web traffic traverses.

We quantify the *exposure* of DNS traffic versus TCP traffic as follows. We begin with Alexa's top 1,000 [9], a list of the 1,000 most popular web sites as estimated by Alexa. For each site, we conducted two experiments. First, we ran a TCP traceroute to the site, targeting port 80 to mimic web traffic. Second, we determined the DNS delegation path for the website's DNS name using the `dig` command's `+trace` feature. The delegation path of a domain name, say www.example.com, is a hierarchy of authoritative DNS servers, such as the authoritative server for .com pointing to the authoritative server for example.com, which in turn points to the authoritative server responsible for www.example.com. We also ran UDP traceroutes to

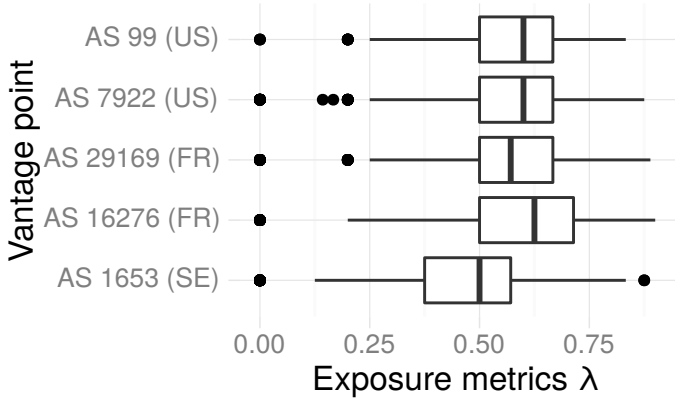


Figure E.2: Five box plots capturing the AS exposure metric λ for Alexa's top 1,000 web sites. The box plots represent five autonomous systems in three countries.

each server in the delegation path, targeting port 53 to mimic DNS resolution.[4] For both experiments, we then mapped all IP addresses in the traceroutes to AS numbers [124], generating both a set of traversed ASes for DNS traceroutes ($\mathcal{D}$) and a set of traversed ASes for web traceroutes ($\mathcal{W}$). Given these two sets for each of Alexa's top 1,000, we compute the fraction of ASes that are *only* traversed for DNS traffic, but *not* for web traffic (λ):

$$\lambda \in [0,1] = \frac{|\mathcal{D} \setminus \mathcal{W}|}{|\mathcal{D} \cup \mathcal{W}|}. \tag{E.1}$$

The metric approaches 1 as the number of ASes that are only traversed for DNS increases. For example, if $\mathcal{D} = \{1,2,3\}$ and $\mathcal{W} = \{2,3,4\}$, then $\lambda = \frac{|\{1,2,3\} \setminus \{2,3,4\}|}{|\{1,2,3\} \cup \{2,3,4\}|} = \frac{|\{1\}|}{|\{1,2,3,4\}|} = \frac{1}{4} = 0.25$. We determined λ for each site in the Alexa top 1,000 from five autonomous systems in three countries.[5] One of our vantage points, the French OVH, is the most popular AS by exit bandwidth as of August 2016. It sees 10.98% of exit traffic, closely followed by AS 12876 (owned by the French Online) that sees 9.33% of exit traffic. Our experiment consisted of 5,000 traceroute runs, 4,773 (95.5%) of which succeeded, and 227 (4.5%) failed.

The result is illustrated in Figure E.2, which shows five box plots capturing λ values for Alexa's top 1,000 sites. The median of all 4,773 λ values is 0.571, so for half of all runs, DNS-only ASes account for 57% or more of all traversed ASes. This result only applies to exit relays that do their own DNS resolution; for relays that use a third-party resolver, the ASes that are traversed between the exit relay and

[4]The tool we developed for this purpose is available online at `https://github.com/NullHypothesis/ddptr`.

[5]The ASes are: OVH (France), Gandi (France), Karlstad University (Sweden), Princeton University (U.S.), and Comcast (U.S.).

its DNS resolver is the metric of interest. We further believe that relays in regions other than Western Europe or North America are likely to witness significantly different exposure of DNS queries because many websites outsource their DNS setup to providers such as Cloudflare whose points of presence are centered around Western Europe and North America. We conclude that adversaries that are unable to observe a Tor user's TCP connection still have many opportunities to see a TCP connection's corresponding DNS request. Such adversaries include (*i*) popular open DNS resolvers such as Google and OpenDNS, (*ii*) DNS root servers, and (*iii*) network adversaries located on the path to the previous two entities.

Determining how Tor exit relays resolve DNS queries

Having shown that the Internet provides ample opportunity for AS-level adversaries to snoop on DNS traffic from exit relays, we now investigate how the exit relays in the Tor network resolve DNS queries in practice. Before this study, we only had anecdotal evidence (*e.g.*, from OpenDNS-powered error messages [142, § 4.1]) that some exit relays would occasionally show.

We identify the DNS resolver of all exit relays by using `exitmap` [141], a scanner for Tor exit relays. `Exitmap` automates running a task such as fetching a webpage over all one thousand exit relays, making it possible to see the Internet through the "eyes" of every single exit relay. Using `exitmap`, we resolve unique, relay-specific domains over each exit relay, to a DNS server under our control. Figure E.3 illustrates this experiment. To improve reliability, we configured `exitmap` to use two-hop circuits instead of the standard three-hop circuits. The first hop was a guard relay under our control. Over each exit relay, we resolved a unique domain PREFIX.tor.nymity.ch. The prefix consisted of the relay's unique 160-bit fingerprint, concatenated to a random 40-bit string whose purpose is to prevent caching, so exit relays indeed resolve each query instead of responding with a cached element. We controlled the authoritative DNS server of tor.nymity.ch, so we could capture both the IP address and packet content of every single query for tor.nymity.ch.

An exit relay can either run its own resolver, as shown in the left exit relay in Figure E.3; or rely on a third-party resolver, such as the one provided by its ISP, as shown in the right exit relay in Figure E.3. If an exit relay runs its own resolver, we expect to receive a DNS request from the exit relay's IP address, but if an exit relay uses a third-party resolver, we expect to receive a request from an unrelated IP address. Having encoded relay-specific fingerprints in the query names, we are able to map queries to exit relays in such cases. We ran this experiment from September 2015 to May 2016, at least once a day.

Figure E.4 illustrates the fraction of DNS requests that four of the most popular organizations could observe. Google averages at 33%, but at times saw more than 40% of all DNS requests exiting the Tor network—an alarming number for a single organization. Second to Google is "Local"—exit relays that run their own resolver, averaging at 12%. Next is OVH, which used to be as popular as local resolvers, but slowly lost its share over time. Note that in contrast to Google, OVH does not run

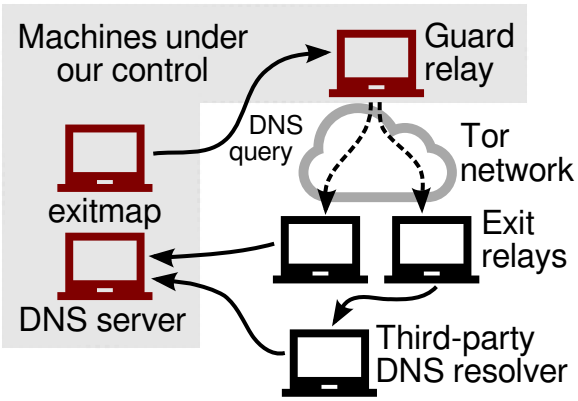


Figure E.3: Our method to identify the DNS resolvers of exit relays. Over each exit relay, we resolve relay-specific domain names that are under our control. By inspecting our DNS server logs, we can then identify the IP address of all exit relay resolvers.

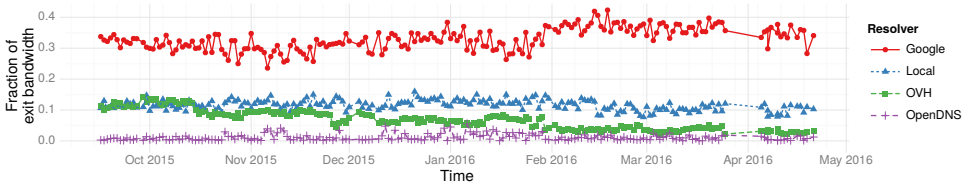


Figure E.4: The popularity of some of the most popular DNS resolvers of exit relays over time. The y axis depicts the fraction of exit bandwidth that the respective resolver is responsible for. Google's DNS resolver is by far the most popular, at times serving more than 40% of all DNS requests coming out of the Tor network. Google is followed by local resolvers, which average at around 12%. Once serving a fair amount of traffic, OVH dropped in popularity, and is now close to OpenDNS, an organization that runs an open resolver.

a public DNS server; the company's resolvers are only accessible to its customers. Finally, there is OpenDNS, which also runs public DNS resolvers. OpenDNS saw occasional spikes in popularity but always remained in the single digits. Apart from the illustrated top resolver setups, the distribution has a long tail, presumably consisting of many ISP resolvers.

DefecTor Attacks

As with conventional correlation attacks, an attacker must observe traffic that is both entering and exiting the Tor network; in contrast to threat models from pre-

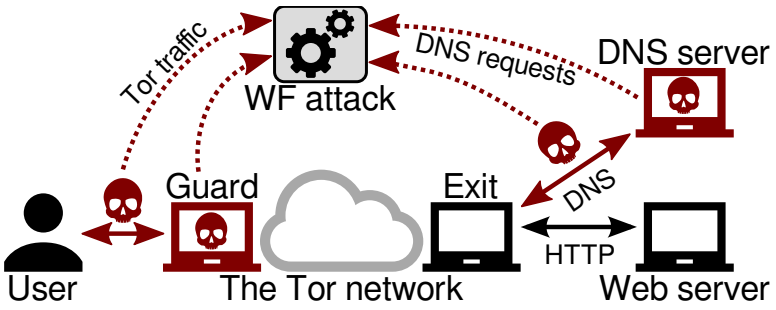


Figure E.5: An overview of the DefecTor attack. An adversary must monitor both ingress (encrypted Tor traffic) and egress (DNS request) traffic. A AS-level adversary between the client and its guard monitors ingress traffic. The same adversary monitors egress traffic between the exit and a DNS server, or the DNS server itself. Both ingress and egress traffic then serve as input to the DefecTor attack.

vious work, we incorporate DNS instead of only TCP traffic. Figure E.5 illustrates our correlation attack; it requires the following building blocks:

- *Ingress sniffing:* An attacker must observe traffic that is entering the Tor network. The attacker can operate on the network level, as a malicious ISP or an intelligence agency. In addition, the attacker can operate on the relay level by running a malicious Tor guard relay. In both cases, the attacker can only observe encrypted data, so packet lengths and directions are the main inputs for website fingerprinting [100].

- *Egress sniffing:* To observe both ends of the communication, an attacker must also observe egress DNS traffic. We expect the adversary either to be on the path between exit relay and a DNS server or to run a malicious DNS resolver or server. We do not expect an attacker to run an exit relay because in this case conventional end-to-end correlation attacks are at least as effective as those we describe here [94].

We combine a conventional website fingerprinting attack operating on traffic from ingress sniffing with DNS traffic observed by egress sniffing, creating DefecTor attacks. Our attacks correlate the web*sites* observed by the website fingerprinting attack in ingress traffic with the web*sites* identified from DNS traffic.[6] Next, we describe how we simulate the DNS traffic from Tor exits, how we map DNS requests to websites, and finally present our two DefecTor attacks.

[6]Our work can be understood as DNS-enhanced traffic correlation attack, *or* as DNS-enhanced website fingerprinting attack.

Approximating DNS traffic from Tor exits

We first investigate the type and volume of DNS traffic that Tor's exit relays send. There are no logs of outgoing traffic from Tor exit relays available to us, and ethical considerations kept us from trying to collect them (*e.g.*, by operating exit relays and recording all the outgoing traffic). We therefore opt to approximate the DNS traffic emerging from Tor exit relays by (*i*) building a model of typical Tor users' website browsing patterns, (*ii*) collecting a minimally invasive dataset of DNS traffic, and (*iii*) accounting for the effects of DNS caching.

Modeling which sites Tor users visit

We first build a model to approximate which websites Tor users visit. As of July 2016, there are about 173 million active websites [97]; the Alexa ranking [9] gives insights into their popularity based on the browsing behavior of a sample of all Internet users. The distribution of the popularity of these websites has previously been fit to a power-law distribution based on the rank of the website [7, 37, 90]. For the pageview numbers of the Alexa top 10,000 websites, we found a power-law distribution to be a good fit as neither a log-normal nor a power-law distribution with exponential cutoff (*i.e.*, a truncated power-law distribution) offered significantly better fits. We used the Python powerlaw package [8] for fitting and picked a power-law distribution with an α parameter of 1.13. When varying the fitting parameter x_{min} that determines beyond which minimum value the power-law behavior should hold in the provided data, we can get different α values. We made a conservative choice of picking this smaller α value as it underestimates the popularity of popular websites and therefore performs worse for the attacker.[7] Thus, we use a power-law distribution to model what websites Tor users visit. On the one hand, this might overestimate the popularity of higher-ranked websites such as Facebook and YouTube because we believe that Tor users—who tend to be privacy-conscious—are more likely to seek out alternatives than the typical Internet user. On the other hand, highly sensitive sites tend to be offered as onion services. We will discuss the implications of our model for browsing behavior later.

Modeling how often Tor users visit each site

Next, we determine how many websites Tor users visit in a certain time span. We approximated this number by setting up an exit relay whose exit policy included only the ports 80 and 443, so our relay would only forward web traffic. We then used the tool `tshark` to capture the timestamps of DNS requests—but no DNS responses. We made sure that our `tshark` filter did not capture packet payloads or headers, so we were unable to learn what websites Tor users were visiting. In

[7] Alexa's page-view numbers ignore multiple visits by the same user on the same day (see `https://support.alexa.com/hc/en-us/articles/200449744`), so the ranking might be slightly off when modeling website visit patterns.

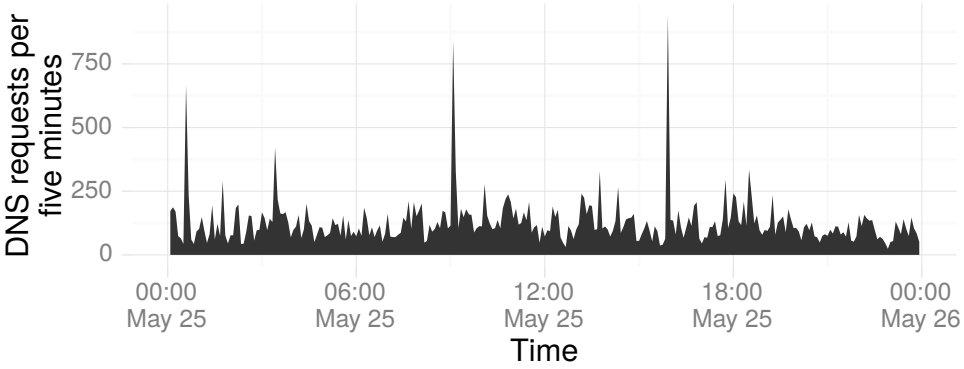


Figure E.6: The number of DNS requests per five-minute interval on our exit relay for May 25, 2016. Using a privacy-preserving measurement method, we only determined approximate timestamps and no content.

addition, we patched `tshark` to log timestamps at a five-minute granularity. The coarse timing granularity allows us to publish this dataset with minimal privacy implications; Section E.8 discusses the ethical implications of this experiment in more detail. We ran the experiment for two weeks, from May 15, 2016 to May 31, 2016, which allowed us to determine the number of DNS requests for 4,832 five-minute intervals. Figure E.6 shows this time series, but for clarity we only plot May 25, 2016. The distribution's median is 105. The time series features several spikes; the most significant one counts 1,410 DNS requests. We repeated the same experiment with the so-called *reduced exit policy*[8] because it contains several dozen more ports and it is more popular among Tor relay operators; as of August 2016, it is used by 7.8% of exit relays by capacity. In comparison, the exit policy containing only port 80 and 443 only accounts for 1.5%. The reduced exit policy resulted in a median of 102 DNS requests per five minutes, so the difference between both policies is only three DNS requests.

We then interpolate these numbers to all Tor exit relays based on their published bandwidth statistics. While we measured a median of 105, the mean of the distribution was 119.3 per five minutes during a two-week period. From DNS statistics of the Alexa top one million websites (see Section E.5) we know that one website visit causes outgoing DNS requests for 10.3 domains on average (assuming a power-law distribution of site popularity as described above, and taking into account Tor's caching of pending DNS requests, ensuring that multiple requests sent by clients for the same domain name only result in one outgoing request by the exit). This means that our exit relay saw an average of 23.2 website visits per ten minutes. Assuming that the two main factors influencing the volume of DNS requests are a

[8]The reduced exit policy is available online at `https://trac.torproject.org/projects/tor/wiki/doc/ReducedExitPolicy`.

relay's *bandwidth* and its *exit policy*, and having shown that the exit policy does not significantly impact the number of DNS requests, we can scale this number up to the whole Tor network using the self-reported bandwidth statistics of exit relays. In particular, we use the bandwidth information reported in the extra-info descriptors that are available on CollecTor [125] and estimate the number of website visits on each of the about 1,200 exit relays active at that time. The resulting average number of websites visited through the Tor network is 288,000 per ten minutes. However, this number is merely an estimate because the interpolation is based on a single exit relay, and the bandwidth data of exit relays is self-reported and might therefore be incorrect.

Recently, Jansen and Johnson measured that the average number of active web (port 80 and 443) circuits in Tor amounts to about 700,000 per ten minutes [72, § 5.3]. Tor Browser, The Tor Project's fork of Firefox, builds one circuit per website entered in the URL bar. How long the circuit remains active depends on Tor Browser settings (primarily `MaxCircuitDirtiness` currently set to ten minutes) and how long TCP streams in the circuit are active: as long as at least one stream is active, the circuit remains active. Each time a new stream is attached to a circuit, the circuit's dirtiness timeout is reset. The number of active circuits serves as an upper bound for the number of websites visited over Tor: visiting different pages of a website will use the same circuit, and visiting a new website will construct a new circuit. Users visiting several pages of a website and websites with long-lived reoccuring connections, like Twitter and Facebook with continuously updating feeds, all lower the number of websites visited in Tor relative to the number of active circuits. For our model we consider the upper bound of 700,000 to be the number of websites visited through the Tor network per ten minutes. This is a conservative choice as more website visits increase the anonymity set of websites possibly visited by a Tor user—and therefore reduces the information an attacker can gain from observed DNS traffic. Later, we revisit the implications of our choice by both scaling the Tor network *up* to ten times its estimated size, and scaling it *down* to the size of 288,000 website visits per ten minutes that we got from our own interpolation described above.

Modeling the effects of DNS caching at Tor exits

To learn what DNS requests the adversary can see, we need to take into account caching of DNS responses. We ignore client-side DNS caching since it is disabled by default, as described in Section E.2. Exit relays, however, do cache DNS requests and we take it into account because all Tor clients using the same exit relay share its cache. In addition to their resolver's cache, exit relays maintain their own DNS cache[9] and enforce a minimum TTL of 60 seconds and a maximum TTL of 30

[9] The code is available online at `https://gitweb.torproject.org/tor.git/tree/src/or/dns.c?id=tor-0.2.9.1-alpha`.

minutes.[10] We refer to this as Tor's *TTL clipping*. However, due to a bug that we identified,[11] the TTL of all DNS responses is set to 60 seconds.

If a Tor client attempts to resolve a domain that an exit relay has cached, the adversary will be unable to observe this request. However, the adversary can record all observed DNS requests over the past x seconds, where x is the maximum TTL value (*i*.e., maintain a sliding window of length x). If a Tor client is attempting to resolve a domain name, the request is either cached or not. If it is not cached, the adversary will see it as a new, outgoing DNS request from the exit relay. If it is cached, it must have been resolved by the exit relay in the last x seconds, and will therefore be in the sliding window. The sliding window technique allows the attacker to capture all relevant DNS requests, regardless of if they are cached or not. We assume that an adversary applies this sliding window technique and models the observable DNS traffic accordingly. The attacker observes a fraction of Tor's exit bandwidth for a specific window length, and together with our website visit frequency estimation, this triggers a number of website visits in our simulation. For each visit event, we randomly draw a website using the power-law website popularity distribution described above and put its DNS requests into the window. As we will see next, we do not need to simulate or consider the fact that the observed fraction of Tor exit bandwidth corresponds to many different exits with individual caches.

Inferring website visits from DNS requests

Given a sliding window full of DNS requests, we investigate how this information can help determine whether a user has visited a website of interest. In April 2016, we visited the Alexa top one million websites five times, and collected all DNS requests that each visit of a website's frontpage generated. We refer to the data collected for one visit as a *sample*. We performed these measurements in five rounds from Karlstad University. Each round browsed all one million websites in random order before visiting the same website again. We used Tor Browser 5.5.4 and configured it *not to browse over Tor*: Tor Browser ensures that the browser behavior is identical to a Tor Browser user over Tor. By not using Tor, we can bypass IP blacklists and CAPTCHAs that Tor users are frequently struggling with. Table E.1 shows the percentage of websites in our dataset that are hosted by Cloudflare or Akamai. We might not be able to access these websites programatically over Tor because they block or filter exit relays, as identified by Khattak *et* al. [81]. We also include Google, which is prevalent in our dataset and restricts access to Tor users for Google's search.

We collected 2,540,941 unique domain names from a total of 60,828,453 DNS requests. The dataset contains 2,260,534 domains that are unique to a particular website, *i*.e., are not embedded on any other top million site; we call these domains *unique domains*. Unique domains are particularly interesting because they reveal

[10] The code is available online at `https://gitweb.torproject.org/tor.git/tree/src/or/dns.c?id=tor-0.2.9.1-alpha#n209`.

[11] The bug report is available online at `https://bugs.torproject.org/19025`.

Table E.1: The percentage of websites in Alexa's top 1 million that use providers that restrict access from Tor [81].

Description	Percentage
Website behind Cloudflare IP address	6.44
Domain on website uses Cloudflare	25.81
Domain on website uses Akamai	33.86
Domain on website uses Google	77.43

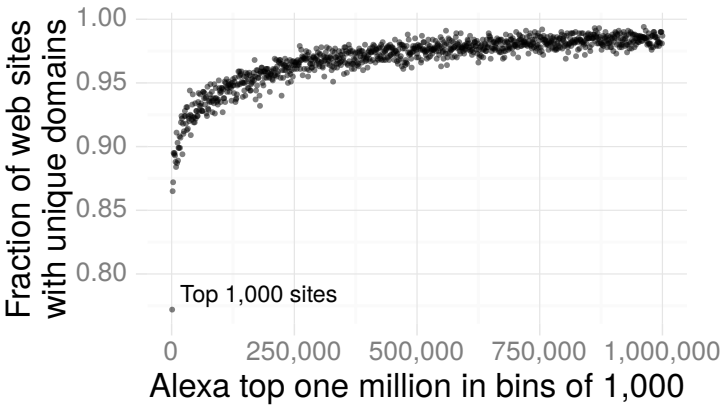


Figure E.7: The fraction of websites in Alexa's top one million that have at least one unique domain. We grouped all domains into 1,000 consecutive, non-overlapping bins of size 1,000. The vast majority of sites (96.8%) have unique domains.

to the adversary what sites among the top million the user has visited. This is not possible for domains such as youtube.com, simply because many websites embed YouTube videos. Figure E.7 shows the fraction of sites with unique domains for websites up to Alexa's top one million. We grouped all domains into 1,000 consecutive, non-overlapping bins of size 1,000. For 96.8% of all sites on the Alexa top one million there exists at least one unique domain. Interestingly, more popular websites are less likely to have a unique domain associated with them: only 77% of the first bin—the most popular 1,000 domains—contain at least one unique domain.

Table E.2 shows summary statistics for the number of domains per website. At least half of the sites have ten domains per website, two of them are unique, suggesting that an adversary can identify many website visits by observing a single unique DNS request.

To evaluate the feasibility of mapping DNS requests to websites, we construct a naïve website classifier that maps the unique domains in a set of DNS requests to the corresponding website that contains a matching set of domains. With five-fold cross-validation on our Alexa top one million dataset (with five samples per site),

Table E.2: Summary statistics for the number of domains per website in the Alexa top 1 million. More than half of the sites embed two domains that are unique to that site.

Domains	**Median**	**Mean ± Stddev**	**Min.**	**Max.**
Per site	10	12.2 ± 11.2	1	397
Unique per site	2	2.3 ± 1.8	0	363

we consider a closed world and an open world. In the closed world, the attacker can use samples from all sites in training; in the open world, some sites are unmonitored and therefore unknown (as per the fold). The closed-world evaluation yields 0.955 recall. In the open-world evaluation, we monitor the Alexa top 500,000 with five samples each and consider 433,000 unmonitored sites. The number of unmonitored sites is determined by our power-law distribution to represent a realistic base rate (for the entire Tor network) for evaluating our classifier: on average, for sites in the Alexa top 500,000 to be visited 2.5 million times, there will be about 433,000 visits to sites outside of Alexa's top 500,000. Our classifier does not take into account the popularity of websites. The open-world evaluation yields a recall of 0.947 for a precision of 0.984. By accounting for request order, per-exit partitioning of DNS requests, TTLs, and website popularity, we expect that classifying website visits from DNS requests can be made even more accurate. Further, a closed world is realistic in our setting: determining the DNS requests made by all 173 million active websites on the Internet is practical, even with modest resources. We use the conservative open world results when simulating the Tor network and the attacker's success in mapping DNS requests to websites. We conclude that for the purpose of identifying websites, observing DNS requests coming out of Tor is almost as effective as observing the web traffic itself.

Classifiers for DefecTor attacks

We extend Wa-kNN from Wang *et* al. [135] (described in Section E.2) by having it take as input a list of sites derived from observing DNS requests. In particular, we implement two DefecTor attacks:

`ctw` We "close the world" on a Wa-kNN classifier that we modified to consider only the distance to observed sites when calculating the k-nearest neighbors. The classifier still considers the distance to all unmonitored sites.

`hp` When Wa-kNN classifies a trace as a monitored site, confirm that we observed the same site in the DNS traffic (ensuring *high precision*). If not, make the final classification unmonitored.

These approaches apply to any website fingerprinting attack. The `ctw` attack increases the effectiveness of conventional website fingerprinting attacks by making

them more akin to a closed-world setting, where websites have known fingerprints and the world is often of limited size. Conceptually, the attack could also include a custom weight-learning run—training only on observed sites—but our initial results showed little to no gain, despite significant increases in testing time. We assume that this is due to the fact that some features of traffic traces are more useful than others, regardless of the training data [67]. The `hp` attack only produces a positive classification if both ingress and egress traffic are consistent, resulting in a simple but effective classifier.

Evaluating DefecTor Attacks

Attack precision and recall

To evaluate our DefecTor attacks, we collected traffic traces in May 2016 using Tor Browser 5.5.4. We modified Tor Browser to not generate network traffic on launch (*i*.e., check for updates, extensions, *e*tc.), and we modified Tor (bundled with Tor Browser) to log incoming and outgoing cells. We then performed 100 downloads for each site in the Alexa top 1,000 and one download for each site in the Alexa top (1k,101k]. We randomly distributed these measurement tasks to a Docker fleet; each download used a fresh circuit without guard relay, and a fresh copy of Tor Browser for up to 60 seconds, in line with the recommendations by Wang and Goldberg [136, § 4]. We cached Tor's network consensus to minimize load on the network. We labeled a measurement as successful if we managed to resolve the domain of the site; we did not prune our dataset further, neglecting issues like Cloudflare CAPTCHAs, outliers, control cells, and localized domains [78]. Presumably, this means that we will underestimate the effectiveness of our attack, but we are primarily interested in the difference between website fingerprinting attacks and DefecTor attacks [136].

We perform ten-fold cross-validation for all of our experiments in the open world setting, monitoring 1,000 sites with 100 instances each, and 100,000 unmonitored sites. The 1:1 ratio between monitored traces and unmonitored traces is to ensure that for the classifier there is equal probability in the testing phase that a trace is a monitored or unmonitored site. In other words, the *base rate* is 0.5 in our experiments. Furthermore, for all experiments we specify the starting Alexa rank of the monitored sites *when simulating sites visited over the Tor network*. We always use the same sample data for website fingerprinting. The popularity of monitored sites is a key factor in the effectiveness of our attacks.

Figure E.8 shows the recall and precision of our DefecTor attacks as a function of the percentage of observed Tor exit bandwidth by the attacker monitoring Alexa sites for sites whose ranks is 10,000 or less. For recall, both `ctw` and `hp` are bound by the percentage of exit bandwidth observed by the attacker (the percentage is an upper bound). It is simply not possible to identify a monitored site in the DNS traffic that the attacker does not see. At 100% of exit bandwidth, `ctw` sees better

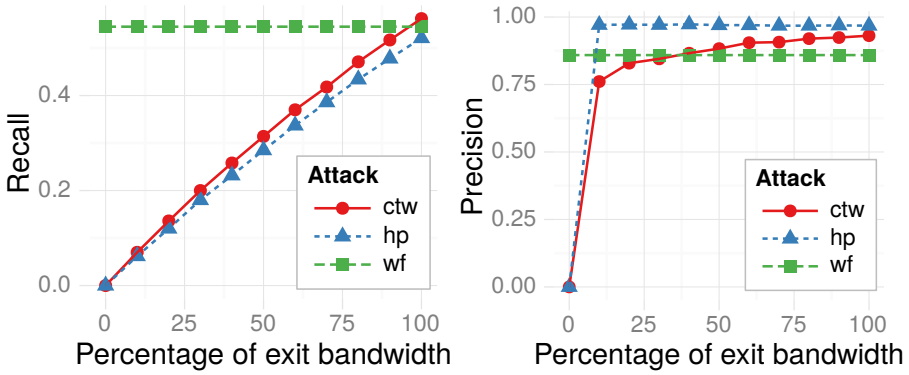


Figure E.8: Recall and precision for an open-world dataset with monitored sites at Alexa rank 10k and lower. We compare our DefecTor attacks (`ctw and hp`) to a conventional website fingerprinting attack (`wf`) for different percentages of observed exit bandwidth.

recall than `wf`. For `hp` the results suggest that:

$$\text{recall}_{\texttt{hp}} = \text{recall}_{\texttt{wf}} * \text{pct}. \tag{E.2}$$

This relationship only holds when observing DNS requests gives a clear advantage to `hp` in terms of precision over `wf` (see the following paragraph). For precision, the `hp` attack has an immediate gain over `wf` as soon as the attacker can observe *a*ny exit bandwidth. Although the `hp` attack has near-perfect precision, the `ctw` attack benefits from observing increasingly more exit traffic, nearly reaching the same levels as `hp` at 100% of the exit bandwidth.

Figure E.9 shows recall and precision at 100% of observed Tor exit bandwidth as a function of the starting Alexa rank of monitored sites (we still monitor 1,000 sites). For popular websites (*i*.e., websites with a high Alexa ranking), there is no difference between our attacks and the `wf` attack. This is because even with a window of only 60 seconds, it is almost certain that at least one user visited any of the most popular sites over Tor. For sites that rank 1,000 or lower (*i*.e., less popular sites), both DefecTor attacks show a clear improvement in precision while `ctw` also shows improved recall—but only at 100% observed exit bandwidth, as shown in Figure E.8. These results paint a bleak picture: an attacker that observes the vast majority of exit bandwidth can use the `ctw` attack as a perfectly precise attack with increased recall over a traditional `wf` attack. On the other hand, an attacker that can observe a small fraction of exit bandwidth can use the `hp` attack as a perfectly precise attack on relatively unpopular sites such as wikileaks.org, which had Alexa rank 10,808 on April 15, 2016. However, Equation E.2 suggests that recall will be low.

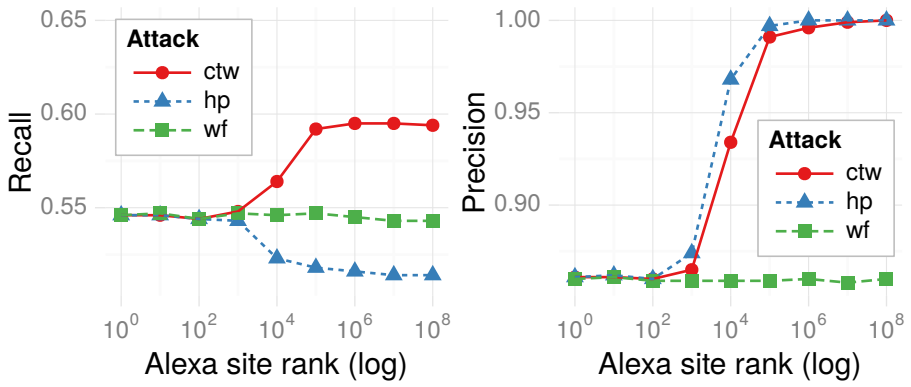


Figure E.9: The recall and precision when varying the starting Alexa rank of monitored sites for 100 percentage of exit bandwidth.

Sensitivity analysis

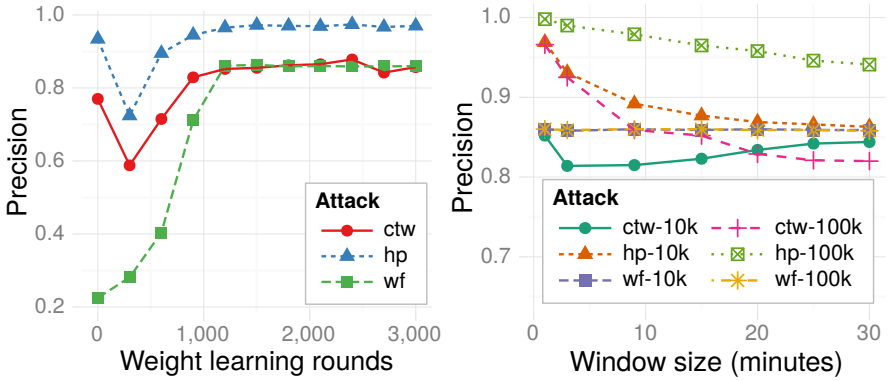
To better understand the extent and limitations of our attacks, we now study the sensitivity of our DefecTor attacks to website fingerprinting defenses, TTL clipping, the growth of the Tor network, and website popularity distribution. In this section, we assume that an adversary can observe Tor exit relays representing 33% of exit bandwidth (as observed on average by Google) and consider only precision (where we see clear gain from both our attacks). Note that the following results largely also apply to weaker attackers that observe a smaller fraction of exit bandwidth for the `hp` attack, but that the `ctw` attack is more sensitive in terms of precision to different bandwidth fractions, as shown above. Unless stated otherwise, we (*i*) perform our evaluation on websites starting from Alexa rank 10,000 upwards, (*ii*) use 2,500 weight-learning rounds, (*iii*) have a 60-second window size, (*iv*) a Tor network scale of 1.0, and (*v*) use the conservative power-law distribution from Section E.5.

Effect of website fingerprinting defenses

The Tor Project is working on a website fingerprinting defense [105]. Most defenses produce bandwidth and/or latency overhead, with a significant increase in overhead as the defense becomes stronger. For example, Juarez *et* al. observe an exponential increase in bandwidth overhead as the protection of the WTF-PAD defense increases [79, § 4.3]. The goal is to find an optimum that provides strong protection while keeping the overhead tolerable for Tor users. To approximate the effect of fingerprinting defenses on DefecTor attacks, we use Wa-kNN with random weights and no weight-learning, which significantly reduces the effectiveness of the attack since some features (like indices of outgoing packets) are several orders of magnitude more useful than others [79].

(a) Estimating the effect of website fingerprinting defenses.

(b) Effect of increasing the analysis time window due to TTL clipping.



(c) Effect of Tor network scale for Alexa ranks 10k and 100k.

(d) Effect of different website popularity distributions.

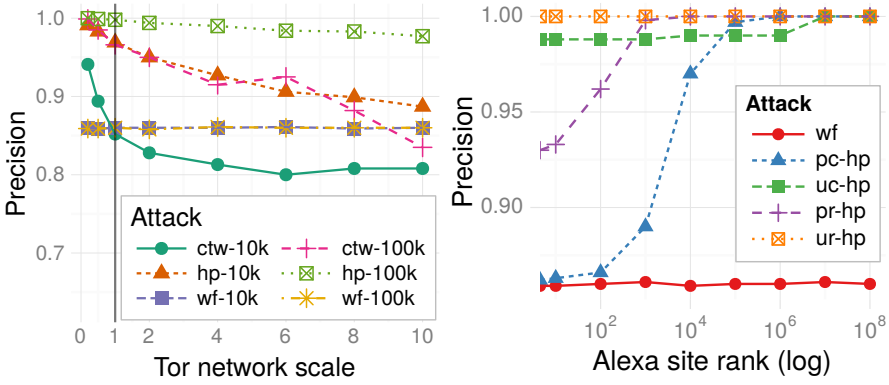


Figure E.10: The effect on attack precision. The defaults are: Alexa from top 10,000, 2,500 weight-learning rounds, 60-second window size, Tor network scale 1.0, and the conservative power-law distribution (pc) with $\alpha = 1.13$.

Figure E.10a shows the effect of weight-learning between 0 and 3,000 rounds. At few to no rounds, the precision for the `wf` attack is below 50%—the classification is more likely to be wrong than right—while there is no impact on the `hp` attack and a relatively small decrease for the `ctw` attack. For recall, which is not shown in the figure, the bound and relationship is as in Equation E.2: for `wf`, at zero rounds, recall is 0.055; for `hp` at zero rounds, recall is 0.019. These results suggest that for website fingerprinting defenses to be effective against DefecTor attacks, the defense must be tuned to cause low recall even if the parameters of website fingerprinting

Table E.3: Median and mean DNS TTL values across Alexa top one million sites. Raw TTLs are unprocessed, as they appear in DNS lookup traces. Tor TTLs adhere to Tor's TTL clipping. Unique refers to the TTLs for unique domains; min unique only considers the unique domains with the minimum TTL for each website.

TTLs	Median TTL (sec)	Mean TTL (sec) ± Stddev
Raw	255	9,780.0 ± 42,930.5
Tor		701.5 ± 755.3
Unique raw	900	13,022.2 ± 35,054.4
Unique Tor		1,005.3 ± 789.6
Min unique raw	60	3,833.9 ± 11,073.6
Min unique Tor		644.2 ± 763.8

attacks are optimized for high recall.

Effect of Tor's TTL clipping

As discussed in Section E.5, due to a bug in Tor, all exit relays cache DNS responses for 60 seconds, regardless of the DNS response's TTL. Therefore, a sliding window covering the last 60 seconds of observed DNS requests suffices to capture all monitored sites through Tor (subject to the fraction of observed Tor exit bandwidth, and mapping DNS requests to sites).

Table E.3 shows the TTL of DNS records in our Alexa top one million dataset from Section E.5 both for the TTL as-is (raw) and when clipped (Tor). We calculate the intended values for TTL clipping, assuming that The Tor Project will fix the aforementioned bug. For each of these cases, we also consider TTLs for all unique domains, and for only the unique domain for each website with the lowest TTL. About half of all sites on Alexa's top one million have a unique domain with a TTL of 60 seconds or less; 48% of the raw unique TTLs are below 60 seconds and only 26% above 30 minutes. Fixing the Tor clipping bug is therefore not sufficient; to mitigate DefecTor attacks, the minimum TTL should be significantly increased. In this case, we find that Tor's TTL clipping has no effect on the median TTL, but significantly reduces the mean TTL.

Suppose that Tor eventually fixes the DNS TTL bug, requiring the attacker to monitor DNS lookups for a time interval equal to the maximum TTL of all unique domains for any monitored site. Figure E.10b shows the effect on precision for different time intervals from 60 seconds to 30 minutes (Tor's `MAX_DNS_ENTRY_AGE` for keeping entries in an exit's DNS resolver cache), and for Alexa starting rank 10,000 and 100,000. For `ctw`, the time interval has a significant effect on both Alexa starting ranks, while `hp` is only affected for sites ranked 10,000 or higher; for less popular sites, the DNS lookup data still significantly improves fingerprinting precision, even with the larger window size.

Effect of Tor network growth

Figure E.10c scales the size of the Tor network with respect to site visits from the estimated status quo to ten times its size, for Alexa starting rank 10,000 and 100,000. At twice its current size, the impact on DefecTor attacks is smaller than increasing the minimum TTL for DNS caching to three minutes, as shown in Figure E.10b. These results indicate that DefecTor attacks will remain practical for many sites in the Alexa top one million, even as the Tor network grows. If we overestimated the current Tor network size in the analysis in Section E.5, our DefecTor attacks would have even higher precision, as shown by the data points to the left of the gray line in Figure E.10c.

Sensitivity to website popularity distribution

To explore the sensitivity of our results to different distributions in how users visit websites, we now evaluate the effectiveness of DefecTor attacks with four different website distributions:

pc A conservative power-law distribution (with $\alpha = 1.13$) that we manually fitted to the Alexa top 10,000 data, slightly underrepresenting the popularity of top Alexa sites. We described this distribution in Section E.5.

pr A realistic power-law distribution (with $\alpha = 1.98$) that is the best fit according to the Python powerlaw library by Alstott *et* al. [8] for the Alexa top 10,000 data.

uc A conservative uniformly random distribution that only considers one million active websites browsed over Tor.

ur A realistic uniformly random distribution that considers 173 million active websites, as reported by Netcraft in July 2016 for the Internet [97].

Figure E.10d shows the effect on the precision of the `hp` attack for the different distributions as we vary the starting Alexa rank. The uniform distributions always have nearly perfect precision. The difference between the two power-law distributions is about one order of magnitude in terms of starting Alexa rank: the realistic distribution gets near perfect at 1,000 and the conservative at 10,000. We conclude that DefecTor attacks are perfectly precise for unpopular sites because it is unlikely that more than one person is browsing a monitored site within the timeframe determined by the window length.

Internet-scale analysis

In the preceding sections we have presented our DefecTor attacks and evaluated their effectiveness, but we have yet to understand what entities can mount them. In this section, we aim to quantify the likelihood that any AS is in a position to mount DefecTor attacks.

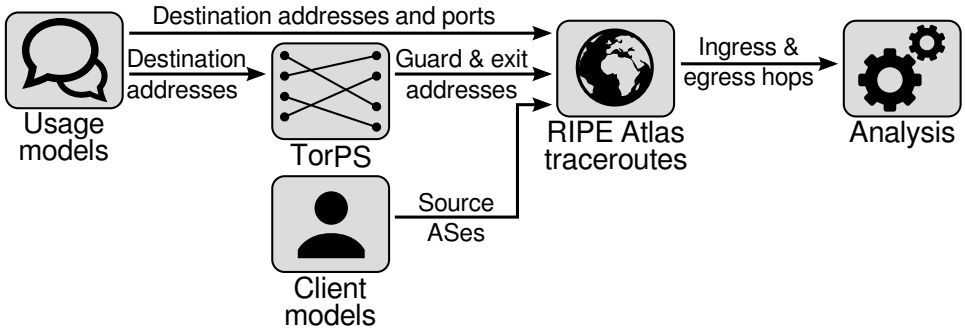


Figure E.11: The relation among our simulation components. Our goal is to determine the ASes a Tor user's traffic traverses into and out of the Tor network. Duplicate ASes on both sides can deanonymize streams.

Approach

Figure E.11 summarizes our simulation approach, which we detail in the next section. In short, we model the activity of Tor users and simulate their path selection using TorPS [75]. TorPS returns guard and exit relays, which we then feed as input—together with source ASes and destination addresses—into our framework that runs traceroutes from RIPE Atlas nodes. The rest of this section describes our approach in detail.

Attack model

We assume that an AS can mount DefecTor attacks if it can see both traffic *entering* the Tor network and DNS traffic *exiting* the Tor network. Recall that an exit relay can perform DNS resolution in two ways; by running a local resolver, or by relying on a third-party resolver, such as its ISP's or Google's public resolver. In the case of exit relays that perform local resolution, an effective position for an attacker is both (*i*) anywhere on the AS path between a Tor client and its guard relay; and (*ii*) anywhere on the path between an exit relay and any of the name servers the exit has to communicate with to resolve a domain. These name servers include the full DNS delegation path, *i*.e., a root name server plus subsequent name servers in the DNS hierarchy. All ASes along the path from the exit relay to the name servers will be able to see the domain names that the exit relay is querying. For exit relays that rely on third-party resolvers, the adversary instead has to be on the path between the exit relay and its DNS resolver.

Simulating Tor user activity with TorPS

To measure the likelihood that an AS can be in a position to perform a DefecTor attack, we use TorPS [75]—short for Tor Path Simulator—which mimics how a

Tor client constructs circuits (see "TorPS" in Figure E.11). TorPS takes as input archived Tor network data [125] and usage models, which are sets of IP addresses that Tor clients talk to—*e*.g. web servers. Given this input, TorPS then simulates for a configurable number of "virtual" Tor clients the way they would select guard and exit relays. TorPS is based on the Tor stable release in version 0.2.4.23. For each simulated client, TorPS uses one guard; this guard selection expires after 270 days. We use TorPS to simulate the behavior of 100,000 Tor clients for the entire month of March 2016.

We need to place our simulated Tor clients into an AS (see "Client models" in Figure E.11). We selected clients in major ISPs in the top-five most popular countries of Tor usage according to Tor Metrics [126]. As of August 2016, the top five countries are the U.S., Russia, Germany, France, and the U.K. For the U.S., we chose Comcast (AS 7922); for Russia, Rostelecom (AS 42610); for Germany, Deutsche Telekom (AS 3320); for France, Orange (AS 3215); and for the U.K., British Telecom (AS 2856).

Having placed simulated Tor clients into ASes, we now model their activity over Tor (see "Usage models" in Figure E.11). We model each client to have visited several websites every day in March 2016.[12] At 9 a.m. EST, the client visits mail.google.com and www.twitter.com. At 12 p.m. EST, the client visits calendar.google.com and docs.google.com. At 3 p.m. EST, the client visits www.facebook.com and www.instagram.com. Finally, at 6 p.m. EST, the client visits www.google.com, www.startpage.com, and www.ixquick.com, and at 6:20 p.m. EST, the client visits www.google.com, www.startpage.com, and www.ixquick.com again. Each of the 100,000 simulated Tor clients thus had $12 \cdot 31 = 372$ opportunities to be compromised given 31 days and 12 site visits per day. TorPS provided a new circuit every ten minutes, regardless of how many distinct connections the client made to different sites; it did not provide a new circuit for different websites if the client visited the group of sites within the same ten-minute window.

For simplicity, we assume that only one DNS request occurs every time a client visits a site. For example, in our model, at 9 a.m. one DNS request will occur for mail.google.com and one DNS request will occur for www.twitter.com. At 6 p.m. three DNS requests will occur, and at 6:20 p.m. those same three DNS requests will occur again. For now, we do not take embedded requests (*i*.e. for embedded website content such as YouTube videos) or caching into account.

Inferring AS-level paths using traceroutes and pyasn

Our Internet-scale analysis also requires learning the AS-level paths from each client to its guard, and from its exit to the destination (see "RIPE Atlas traceroutes" in Figure E.11). We decided against the commonly applied AS path inference because Juen *et* al. showed that it can be quite inaccurate [80]. Traceroutes, in contrast,

[12] We modeled our client behavior off of the "Typical" model that Johnson *et* al. [76, § 5.1.2] used.

Table E.4: The coverage of RIPE Atlas nodes that are co-located with Tor guard and exit relays as of May 2016.

Atlas probe coverage	**Tor guard ASes**	**Tor exit ASes**
By bandwidth	73.59%	57.53%
By number	50.69%	52.25%

yield significantly more accurate paths, but are difficult to run from Tor relays: Past work involved asking relay operators to run traceroutes on behalf of the researchers [80, § 4]. This approach yielded traceroutes from relays representing 26% of exit bandwidth, but does not scale well. Instead of running traceroutes from Tor relays, we leverage the RIPE Atlas [111] platform, a volunteer-run network measurement platform consisting of thousands of lightweight and geographically spread *probes* that can be used as vantage points for traceroutes. Our key observation is that RIPE Atlas has probes in many ASes that also have Tor relays. We leverage this observation by designing measurements to run traceroutes from Atlas probes that are located in the same AS as exit relays, to each of the destinations in question.

Table E.4 shows that for a day in May 2016, we found that RIPE Atlas had probes in 52% of ASes that contain exit relays, and in 51% of ASes that contain Tor guard relays. More importantly, we found that Atlas ASes cover 58% of exit *bandwidth* and 74% of guard *bandwidth*. This statistic is important given that Tor clients select relays weighted by their bandwidth, and the bandwidth of Tor relays is not uniformly distributed. Given the growth of both Tor and Atlas, we expect these numbers to increase in the future. In addition to Atlas, we also considered using PlanetLab [1] to initiate traceroutes, but unfortunately most PlanetLab nodes are located in research and education networks [16], and are thus not suited for performing our measurements.

We performed traceroutes from the five Tor client ASes outlined above to all their respective guard relay IP addresses that TorPS determined. To measure the paths from exit relays to their DNS resolvers, we performed the following traceroutes, simulating four different DNS configurations:

- *ISP DNS:* To investigate the scenario in which an exit relay uses its ISP's resolver, we chose to represent this as the resolver being in the same AS as the exit relay. Thus, no traceroutes were necessary for this experiment. We acknowledge that this is not necessarily the case, but assume that it holds for the majority of exit relays.

- *Google DNS:* This scenario represents an exit relay using Google's public resolver. To measure the AS path, we perform traceroutes from a RIPE Atlas node in the AS of the exit relay to Google's public DNS resolver, *i*.e., 8.8.8.8.

- *Local DNS:* To measure the paths that would be traversed if an exit relay were running its own, local resolver (*e.g.*, the popular service `unbound`), we used the command line tool `dig` with the `+trace` option to determine the iterative resolution process. We tracked all name server IP addresses from referrals at each level of the delegation path, and performed traceroutes to those IP addresses.

- *Status quo:* This scenario represents the state of the Tor network as of March 2016, a combination of the above three configurations. Recall that in Section E.4, we determined the IP addresses of the resolvers that exit relays use. We ran traceroutes to these very IP addresses. For the exit relays that used several resolvers during March, we randomly assigned one to the relay. We ended up having data for 73% of the exit relays that TorPS ended up picking.[13]

We then mapped each IP address in every traceroute to its corresponding AS (see "Analysis" in Figure E.11). The Python module `pyasn` [11] relies on BGP routing tables to perform these mappings; by using a routing table that coincides with the time when we performed our traceroutes, we can obtain accurate AS-level mappings. This method is subject to inaccuracies due to BGP route hijacks or leaks, but we expect those events to be relatively unlikely for the time period and IP prefixes that we are concerned with.

Putting it all together

We consider the same two security metrics that Johnson *et* al. [76, § 4.2] originally proposed; we aim to estimate (*i*) the fraction of compromised streams per simulated Tor user, and (*ii*) the amount of time it would take for the first compromise to occur. For both metrics, we consider the four DNS configurations outlined above. Our simulation can reveal the respective average threat that a given DNS configuration poses for Tor users.

Each traceroutes run yielded two sets of ASes, one from the Tor clients' ASes to their guard relays, and one from approximately half of the exit relays' ASes to the different destinations, which depend on the exit relays' DNS configurations. We intersect both AS sets (the "ingress" and "egress" hops of Figure E.11) and classify a website visit as compromised if the intersection is non-empty. As stated earlier, for some exit relays we did not have associated AS-level paths to a particular destination, either due to a lack of co-located RIPE Atlas probes, or because of missing traceroute information. In these cases, we checked if the exit AS had the potential to launch an attack by itself, and if not, we labelled the stream as uncompromised to err on the conservative side.

[13]The missing 27% are due to the churn in exit relays. Since we did not run our `exitmap` experiment each hour, we were bound to miss some exit relays.

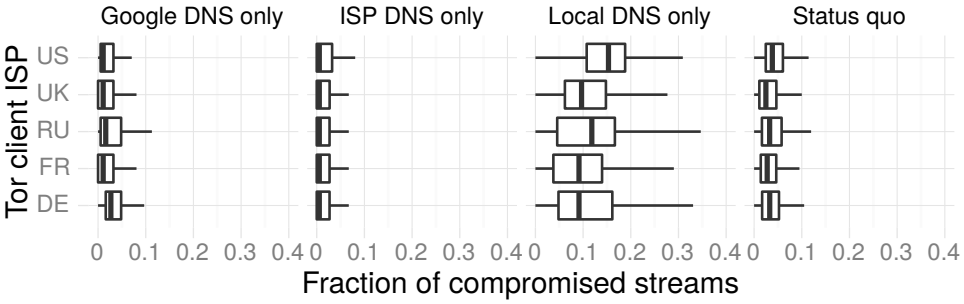
To compute the fraction of compromised streams, we counted the streams that were compromised for every simulated user out of a possible maximum of 372. To compute the time until first compromise, we determined the first stream in which the user was compromised, took its timestamp, and calculated the offset from the beginning of March 1, 2016. For users that were not compromised during the month of March, we assigned the maximum value of 31 days as the time until first compromise, which is reflected in the plots in our next section. Users who were compromised immediately would have a value of 0, signifying that they were compromised at the very beginning of March 1.

Results

Figures E.12a and E.12b illustrate our results as box plots. Each figure contains four subfigures, one for each DNS configuration. Each box plot contains five rows, one for each Tor client AS. For clarity, we did not plot any outliers beyond the box whiskers. For the fraction of compromised streams, an ideal setup has its median at 0. For the time until first compromise, an ideal median is 31. Both figures show that the "ISP DNS only" setup is the safest for Tor users, *i*.e., it exhibits on average the least number of compromised streams while also on average counting the most days until compromise. This setup is closely followed by "Google DNS only," the status quo, and finally "Local DNS only," which fares worse than all other setups. We expected "ISP DNS only" to do best because if all exit relays use their ISP's resolvers, there is only one AS to contend with on the egress side—the exit relay's. The Google setup fares similarly well; most likely because of Google's heavily anycast infrastructure which minimizes the number of AS hops. The status quo does significantly better than the "Local DNS" results, presumably because only around 12% of Tor exit relays actually do their own resolution. The large variance observed in Figure E.12b for "ISP DNS" and "Google DNS" is due to using 31 days as a placeholder for simulated clients who were never compromised. However, a safe configuration against AS-level adversaries, which our figures capture, is not necessarily the best setup for Tor users. For example, ISP-provided DNS resolvers can be misconfigured, subject to censorship, or simply be a forwarder to Google's resolver, which already serves numerous exit relays and whose centralization poses a threat to the anonymity of Tor users. We will explore this trade-off in greater detail in Section E.8.

Interestingly, we find differences in our five client ASes. These differences are particularly striking in Figure E.12b. For "Google DNS only," the median time until compromise differs by around seven days between DE and UK, and around eight days between DE and FR. For "ISP DNS only," the median time until compromise differs by around six days between US and DE, and around five days between US and FR. Also, we notice that DE fares worse than the others in the "Google DNS only" scenario and better than the others in the "ISP DNS only" scenario. We conclude that the location of Tor clients matters and should be considered in future traffic correlation studies.

(a) The fraction of compromised streams of simulated Tor clients.



(b) The time until simulated Tor clients got first compromised.

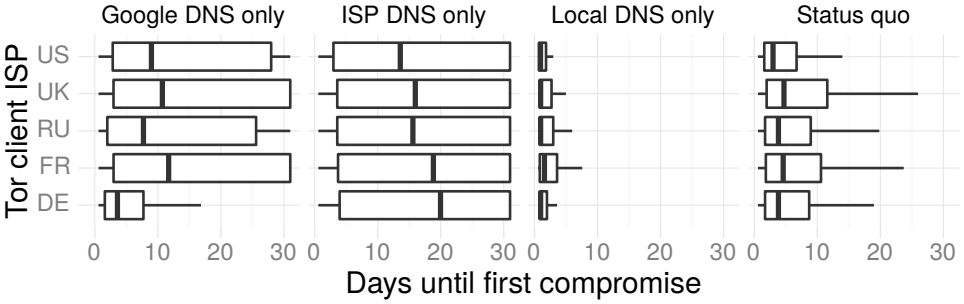


Figure E.12: The fraction of compromised streams and the time until first compromise for our simulated Tor clients. We placed these clients in five popular client ASes in the U.S., the U.K., Russia, France, and Germany. For exit relays, we consider the status quo (on the very right) plus three hypothetical DNS configurations for all exit relays. We do not plot outliers beyond the box plots' whiskers. In both experiments, the safest configuration is "ISP DNS only," *i.*e. have all exit relays use their ISP's DNS resolver.

Discussion

In this section, we briefly discuss the ethics of our research and ways to defend against DefecTor attacks.

Ethical considerations

In Section E.5, we discussed setting up an exit relay to determine the number of DNS requests per five minute interval. Since our exit relay was forwarding traffic of Tor users, we contacted Princeton University's institutional review board (IRB) before running the experiment. Our IRB deemed that this research did not fall

within the realm of human subjects research. In addition to contacting our IRB, we adhered to The Tor Project's ethics guidelines [83]. Specifically, (*i*) we ensured that we only collected data that is safe to publish, (*ii*) we only collected data we needed, and (*iii*) we limited the granularity of the data to minimize the likelihood of reidentification. The risk to Tor users of this experiment is negligible. As for the benefits, by conducting this experiment, we can improve our understanding of the risks that DNS poses to the anonymity of Tor users and use this understanding to improve protection for Tor users in the future. Thus, we believe that the benefits of our experiment outweigh the risks.

Defending against DefecTor attacks

We now discuss ways to defend against DefecTor attacks. We distinguish between short-term solutions that can be implemented quickly (Section E.8), and long-term solutions that need significantly more work (Section E.8). Our discussion of countermeasures is not comprehensive, and we defer a more detailed analysis to future work.

Short-term solutions

Exit relay operators face a dilemma: they must either operate their own resolver, which exposes DNS queries to network adversaries; or, they must use a third-party DNS resolver, which exposes DNS queries to a third party. Clearly, the goal is to minimize exposure of DNS requests, but there are several dimensions to this. In lieu of substantial DNS protocol improvements, we envision three extreme design points, in which *all* exit relays use (*i*) Google's DNS resolver; (*ii*) their own, local resolver; or (*iii*) the resolver provided by their ISP.

If all exit relays were to use Google's public resolver, the company would obtain metadata about the activity of all Tor users, which runs counter to Tor's design goal of distributing trust. We clearly should avoid this scenario. Fifield *et* al.'s [54] censorship circumvention system meek used to use Google's cloud infrastructure to tunnel the traffic of censored users up until May 2016 [53]. While the system was operational, thousands of meek clients selected exit relays that use Google's public resolver, which means that Google saw both traffic entering and, partially, exiting the Tor network, allowing the company to mount DefecTor attacks. Next, consider a Tor network that only uses local resolvers. In this case, Tor is fully independent of third-party resolvers, at the cost of each iterative DNS query being exposed to a diverse set of ASes in the network, allowing several parties to learn the DNS queries of Tor users. Finally, all exit relays could simply use their ISP-provided resolver. This would minimize the network exposure of DNS requests as resolvers are frequently in the same AS as exit relays, and AS-level adversaries would be unable to distinguish between DNS requests from exit relays and unrelated ISP customers. However, this setup introduces the possibility of misconfigured and censored DNS resolvers [142, § 4.1]. Besides, just a few ASes—OVH, for example—

host a disproportionate amount of exit relays, turning them into the very centralized data sinks that Tor aims to avoid.

Considering the above, we believe that exit relay operators should avoid public resolvers such as Google and OpenDNS. Instead, they should either use the resolvers provided by their ISP, or run their own, particularly if the operator's ISP already hosts many other exit relays. Local resolvers can further be configured to minimize information leakage, by enabling QNAME minimization [24]. There likely is a measurable performance difference between a local resolver and Google's resolver, but we believe that this difference pales in comparison to other performance issues in Tor such as head-of-line blocking.

Finally, Tor can fix the Tor clipping bug we discovered and consider significantly increasing the minimum TTL for the DNS cache at exit relays to make DefecTor attacks less precise. This adjustment requires finding the longest acceptable TTL that does not have a notable negative detriment to user experience. Further, as soon as the clipping bug is fixed, website operators of sensitive websites can opt to increase the TTL of their DNS records.

Long-term solutions

Additional practical defenses are on the horizon. Zhu *et* al. [147] proposed T-DNS, which employs several TCP optimizations to transport the DNS protocol over TLS and TCP. The TLS layer provides confidentiality between exit relays and their resolvers. Finally, site operators whose users are particularly concerned about safety should offer an onion service as an alternative. Facebook, for example, set up facebookcorewwwi.onion. When connecting to the onion service, Tor users never leave the Tor network, and hence do not need DNS—as long as the onion service does not embed non-onion service content.

Deploying defenses against website fingerprinting attacks in Tor should be an important long-term goal, as well. Although growing the Tor network will help defend against DefecTor attacks to some degree, the most important change is to deploy defenses against these attacks. Since DefecTor attacks significantly increase precision of website fingerprinting attacks, defenses should be designed to significantly reduce the recall of website fingerprinting attacks, even when the website fingerprinting attack is configured to sacrifice precision for recall.

Conclusion

In this paper, we have demonstrated how AS-level adversaries can use DNS traffic from Tor exit relays to launch more effective website fingerprinting attacks, to learn what websites Tor users are visiting. Mapping DNS traffic to websites is highly accurate even with simple techniques, and improves the precision when monitoring relatively unpopular websites. We further developed a method to identify the DNS resolver for each Tor exit relay, and found that a set of exit relays comprising 40% of all Tor exit relay bandwidth use the Google public DNS servers. Although

this concentration of DNS query traffic reduces the expanse of ASes that can see DNS query traffic emanating from exit nodes, this configuration nonetheless gives a single administrative entity considerable visibility into the traffic that is exiting the Tor network. Tor relay operators should take steps to ensure that the network maintains more diversity into how exit relays resolve DNS domains. To mitigate the risk of website fingerprinting attacks in light of our work, we suggest that local DNS resolvers on Tor exit relays implement privacy-preserving techniques such as DNS QNAME minimization, which minimizes the amount of information about the domain name that each iterative query contains. We publish all our code, data, and replication instructions on our project page, which is available online at `https://nymity.ch/tor-dns/`.

Acknowledgments

We want to thank Jedidiah R. Crandall for providing infrastructure for our measurements, Aaron Johnson for help with TorPS, Robayet Nasim for running DefecTor experiments, and Tom Ritter and the anonymous reviewers for providing helpful feedback. This research was supported in part by the Swedish Foundation for Strategic Research grant SSF FFL09-0086; the Swedish Research Council grant VR 2009-3793; the Swedish Internet Fund grant "Hoppet till Tor"; the National Science Foundation Awards CNS-1540055 and CNS-1602399; and the Center for Information Technology Policy at Princeton University.

5. Concluding Remarks

This thesis addressed the questions how Online Social Networks can be decentralized to improve user privacy and what new privacy issues emerge in Decentralized Online Social Networks and other decentralized systems. It contributes to the research in the field by designing several protocols for OSN functionalities that do not rely on central system components and by analyzing the threats to user privacy caused by metadata. It also presents a concrete attack based on metadata from a deployed decentralized system, the Tor anonymization network, and discusses how the attack can be mitigated to better preserve user anonymity.

From the security and privacy analysis in Article A we conclude that DOSNs, while bearing a great potential for improving user privacy, have to be implemented carefully not to reveal sensitive user data that had been protected in a centralized system with a trusted provider. New adversaries or those that had limited capabilities in a centralized setting become relevant in the DOSN context. Only encrypting the content is not enough to protect user privacy. The decentralized architectures, based on P2P networks and DHTs, exhibit more diverse points of attack such as inferences from encrypted storage objects, encryption headers, key distribution mechanisms or traffic analysis. They might leak more sensitive metadata information than what was exposed to external attackers in centralized systems. We listed some general techniques to mitigate these privacy threats. Some of the possible attacks are straightforward to protect from, while other information leaks are harder to avoid and protection techniques depend on the concrete DOSN implementation.

In Articles B, C and D, we presented implementations for OSN functionalities in decentralized systems that take these new threats into account. In our work on password authentication features for P2P systems, presented in Article B, we show how the well-known username–password paradigm for authentication can be transferred to systems without any central components. It allows user logins on multiple devices and shows how password management tasks, such as password change, blocking of lost devices or password recovery can be implemented in a user-friendly and secure way. The implementations entail several usability–security trade-offs that need to be re-evaluated individually for each system where the password authentication feature is to be deployed. Also the implementation proposal

for a user search feature discussed in Article C has to balance usability and privacy requirements. Using a knowledge threshold, it allows legitimate users to find others while adversaries that do not possess enough information about the target user do not learn any information about them. Our evaluation results suggest that this might be an effective protection at least against crawling attacks. The work on event invitations for decentralized systems in Article D presents a possible implementation of this functionality, featuring fine-grained privacy settings. It also introduces several techniques that might be useful in other contexts, such as a "commit-disclose" protocol that allows to share private data only with users who give a certain commitment, or a "controlled ciphertext inference" protocol that allows eligible users to derive the size but not the content of a datastructure.

In contrast to the other included articles, Article E focuses on a concrete attack of a specific deployed system, the Tor anonymization network. The website-fingerprinting-attack that is discussed in this work illustrates the larger attack surface of decentralized systems: traffic analysis of encrypted connections can leak sensitive data. This can be seen as one instance of metadata leakage, as discussed in Article A even if the investigated system is not a DOSN. The evaluation of the vulnerability illustrates the power of an adversary that can combine different sources of information – encrypted ingress traffic with outgoing DNS requests in this case. The analysis of different countermeasures shows that further decentralization – of the DNS resolution in this case – can be part of the solution.

There are several possible directions for future work. One possible path to proceed on is to explore the OSN functionalities we have looked at in more depth. The work presented on password login mechanisms for a decentralized system leaves several open problems, such as formal security proofs of the protocol properties and protection against offline password guessing attacks. To achieve the latter, distributed password verification [32, 33] or other password-hardening techniques could be incorporated into the protocol. Using these techniques, no single node would store information that could be used to mount brute-force attacks against a user password. Only if a certain amount of nodes collude or get compromised, offline attacks would be possible. For the user search protocols, it might be interesting to combine the two different implementation approaches described in the article and for the event invitation feature, extensions like transferable invites, where invited users can invite other users, but only to a certain degree of indirection, might be interesting to explore. For the analysis of the attack on the Tor network it would be interesting to quantify different proposed countermeasures, for example the effect of query name minimization and to investigate how well the attack works in more challenging settings such as multi-tab browsing, that are known to be hard for classical website fingerprinting attacks.

Another possible path for future work is to look at other OSN functionalities in the same way as we did for password management, user search and event invitations. There might be similar challenges and interesting problems to solve for features such as a friend activity feeds, direct messaging or group chats in the context of a DOSN where no trusted third party is available.

Finally, it might be interesting to look at interaction effects on implementation constraints and privacy guarantees when combining the functionalities that we have looked at in isolation, into one comprehensive DOSN system. This might for example exacerbate the metadata privacy problem outlined in Article A and require new protection mechanisms.

The work in this thesis only touches on some of the technical aspects of the complex endeavor of building a full-featured and privacy-preserving DOSN. But we hope that the metadata privacy analysis, the feature implementation proposals and the attack analysis are building blocks that are not only helpful in DOSN research but applicable to other contexts of decentralized systems as well.

6. Bibliography

[1] PlanetLab – an open platform for developing, deploying, and accessing planetary-scale services. URL `https://www.planet-lab.org`. (cited on p. 125)

[2] S. M. A. Abbas, Johan A. Pouwelse, Dick H. J. Epema, and Henk J. Sips. A gossip-based distributed social networking system. In Sumitra Reddy, editor, *WETICE*, pages 93–98. IEEE Computer Society, 2009. (cited on pp. 44 and 45)

[3] Karl Aberer, Anwitaman Datta, and Manfred Hauswirth. Efficient, self-contained handling of identity in peer-to-peer systems. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):858–869, 2004. (cited on p. 45)

[4] Philip E. Agre. P2P and the promise of internet equality. *Communications of the ACM*, 46(2):39–42, 2003. (cited on p. 2)

[5] Philip E. Agre and Marc Rotenberg. *Technology and Privacy: The New Landscape*. MIT Press, 1997. ISBN 9780262511018. (cited on p. 5)

[6] Luca Maria Aiello and Giancarlo Ruffo. Secure and flexible framework for decentralized social network services. *IEEE PERCOM*, pages 594–599, 2010. (cited on pp. 12 and 31)

[7] Kamal Ali and Mark Scarr. Robust methodologies for modeling web click distributions. In *WWW*. ACM, 2007. URL `https://nymity.ch/tor-dns/pdf/Ali2007a.pdf`. (cited on p. 111)

[8] Jeff Alstott, Ed Bullmore, and Dietmar Plenz. powerlaw: A Python package for analysis of heavy-tailed distributions. *PLoS ONE*, 9(1), 2014. URL `https://nymity.ch/tor-dns/pdf/Alstott2014a.pdf`. (cited on pp. 111 and 122)

[9] Amazon Web Services. Alexa top sites. URL `https://aws.amazon.com/alexa-top-sites/`. (cited on pp. 106 and 111)

[10] Jonathan Anderson, Claudia Diaz, Joseph Bonneau, and Frank Stajano. Privacy-enabling social networking over untrusted networks. In *WOSN*. ACM, 2009. (cited on pp. 9, 31, and 39)

[11] Hadi Asghari. pyasn – Python IP address to autonomous system number lookup module. URL `https://github.com/hadiasghari/pyasn`. (cited on p. 126)

[12] Michael Backes, Marek Hamerlik, Alessandro Linari, Matteo Maffei, Christos Tryfonopoulos, and Gerhard Weikum. *Anonymity and Censorship Resistance in Unstructured Overlay Networks*, page 147–164. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Nov 2009. ISBN 978-3-642-05147-0. (cited on p. 2)

[13] Michael Backes, Matteo Maffei, and Kim Pecina. *A Security API for Distributed Social Networks.*, volume 11, page 35–51. 2011. URL `http://staging.www.isocdev.org/sites/default/files/backes.pdf`. (cited on p. 16)

[14] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: an online social network with user-defined privacy. *SIGCOMM Comput. Commun. Rev.*, 39:135–146, 2009. ISSN 0146-4833. (cited on pp. 9, 12, 31, 40, and 86)

[15] Xiao Bai, Marin Bertier, Rachid Guerraoui, Anne-Marie Kermarrec, and Vincent Leroy. Gossiping personalized queries. In Ioana Manolescu, Stefano Spaccapietra, Jens Teubner, Masaru Kitsuregawa, Alain Léger, Felix Naumann, Anastasia Ailamaki, and Fatma Özcan, editors, *EDBT*, volume 426 of *ACM International Conference Proceeding Series*, pages 87–98. ACM, 2010. ISBN 978-1-60558-945-9. (cited on p. 68)

[16] Suman Banerjee, Timothy G. Griffin, and Marcelo Pias. The interdomain connectivity of PlanetLab nodes. In *PAM*. Springer, 2004. URL `https://nymity.ch/tor-dns/pdf/Banerjee2004a.pdf`. (cited on p. 125)

[17] Salman Baset and Henning Schulzrinne. An analysis of the Skype peer-to-peer internet telephony protocol. *CoRR*, abs/cs/0412017, 2004. (cited on p. 68)

[18] Matthias Bender, Sebastian Michel, Peter Triantafillou, Gerhard Weikum, and Christian Zimmer. MINERVA: Collaborative P2P search. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *VLDB*, pages 1263–1266. ACM, 2005. ISBN 1-59593-154-6, 1-59593-177-5. (cited on p. 68)

[19] Krista Bennett, Christian Grothoff, Tzvetan Horozov, and J. T. Lindgren. An encoding for censorship-resistant sharing, 2003. URL `https://gnunet.org/svn/GNUnet-docs/WWW/download/ecrs.pdf`. (cited on pp. 46 and 48)

[20] Krista Bennett, Christian Grothoff, Tzvetan Horozov, and Ioana Patrascu. Efficient sharing of encrypted data. In Lynn Margaret Batten and Jennifer Seberry, editors, *ACISP*, volume 2384 of *Lecture Notes in Computer Science*, pages 107–120. Springer, 2002. ISBN 3-540-43861-0. (cited on p. 46)

[21] Stevens Le Blond, Zhang Chao, Arnaud Legout, Keith Ross, and Walid Dabbous. I know where you are and what you are sharing: Exploiting P2P communications to invade users' privacy. In *Internet Measurement Conf.*, 2011. (cited on p. 38)

[22] Oleksandr Bodriagov and Sonja Buchegger. Encryption for peer-to-peer social networks. In *IEEE SPSN*, 2011. (cited on pp. 12 and 31)

[23] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: Byzantine resilient random membership sampling. *Computer Networks*, 53(13):2340–2359, 2009. (cited on pp. 46 and 60)

[24] Stephane Bortzmeyer. RFC 7816 – DNS query name minimisation to improve privacy, March 2016. URL `https://tools.ietf.org/html/rfc7816`. (cited on p. 130)

[25] John G. Brainard, Ari Juels, Ronald L. Rivest, Michael Szydlo, and Moti Yung. Fourth-factor authentication: somebody you know. In *CCS*, pages 168–178. ACM, 2006. (cited on p. 54)

[26] S. Buchegger and A. Datta. A case for P2P infrastructure for social networks - opportunities & challenges. In *Sixth International Conference on Wireless On-Demand Network Systems and Services, 2009. WONS 2009*, pages 161–168, February 2009. (cited on p. 9)

[27] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. PeerSoN: P2P social networking: early experiences and insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, SNS '09, pages 46–52, New York, New York, USA, 2009. ACM Press. (cited on pp. 9, 10, and 31)

[28] Dominik Buszko, Wei-Hsing (Dan) Lee, and Abdelsalam Helal. Decentralized ad-hoc groupware API and framework for mobile collaboration. In *GROUP*, pages 5–14. ACM, 2001. (cited on p. 85)

[29] Xiang Cai, Rishab Nithyanand, and Rob Johnson. CS-BuFLO: A congestion sensitive website fingerprinting defense. In *WPES*. ACM, 2014. URL `https://nymity.ch/tor-dns/pdf/Cai2014a.pdf`. (cited on p. 103)

[30] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *CCS*. ACM, 2014. URL `https://nymity.ch/tor-dns/pdf/Cai2014b.pdf`. (cited on p. 105)

[31] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *CCS*. ACM, 2012. URL `https://nymity.ch/tor-dns/pdf/Cai2012a.pdf`. (cited on p. 105)

[32] Jan Camenisch, Anja Lehmann, Anna Lysyanskaya, and Gregory Neven. *Memento: How to Reconstruct Your Secrets from a Single Password in a Hostile Environment*, pages 256–275. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Aug 2014. ISBN 978-3-662-44380-4. (cited on p. 134)

[33] Jan Camenisch, Anja Lehmann, and Gregory Neven. *Optimal Distributed Password Verification*, pages 182–194. ACM Press, 2015. ISBN 978-1-4503-3832-5. (cited on p. 134)

[34] Monica Chew, Dirk Balfanz, and Ben Laurei. (Under)mining privacy in social networks. In *IEEE W2SP*, 2008. (cited on p. 32)

[35] Shihabur Rahman Chowdhury, Arup Raton Roy, Maheen Shaikh, and Khuzaima Daudjee. A taxonomy of decentralized online social networks. *Peer-to-Peer Networking and Applications*, 8(3):367–383, May 2014. ISSN 1936-6442, 1936-6450. (cited on p. 9)

[36] I. Clarke, O. Sandberg, M. Toseland, and V. Verendel. Private communication through a network of trusted connections: The dark freenet, 2010. URL `https://freenetproject.org/papers/freenet-0.7.5-paper.pdf`. (cited on pp. 46 and 48)

[37] Aaron Clauset, Cosma Rohilla Shalizi, and Mark E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4), 2009. URL `https://arxiv.org/pdf/0706.1062`. (cited on p. 111)

[38] Landon P. Cox, Christopher D. Murray, and Brian D. Noble. Pastiche: Making backup cheap and easy. In *OSDI*. USENIX Association, 2002. (cited on p. 45)

[39] Leucio Antonio Cutillo, Refik Molva, and Thorsten Strufe. Safebook: A privacy-preserving online social network leveraging on real-life trust. *IEEE Communications Magazine*, 47(12):94–101, 2009. (cited on pp. 9, 12, 31, 44, 45, 61, and 86)

[40] George Danezis and Richard Clayton. Introducing traffic analysis. In Allessandro Acquisti, Stefanos Gritzalis, Costas Lambrinoudakis, and Sabrina De Capitani di Vimercati, editors, *Digital Privacy: Theory, Technologies, and Practices*, chapter 5, pages 95–116. Auerbach, 2007. ISBN 978-1-4200-5271-6. (cited on pp. 5, 31, and 32)

[41] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Security & Privacy*. IEEE,

2003. URL `https://nymity.ch/tor-dns/pdf/Danezis2003a.pdf`. (cited on p. 100)

[42] Judith DeCew. Privacy. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2015 edition, 2015. (cited on p. 5)

[43] S. Demeyer. Research methods in computer science. *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, page 600, Sep 2011. (cited on p. 20)

[44] Roger Dingledine and Nick Mathewson. Tor protocol specification. URL `https://spec.torproject.org/tor-spec`. (cited on p. 104)

[45] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *USENIX Security*. USENIX, 2004. URL `https://nymity.ch/tor-dns/pdf/Dingledine2004a.pdf`. (cited on pp. 100 and 104)

[46] John R. Douceur. The Sybil attack. In *IPTPS*, volume 2429, pages 251–260. Springer, 2002. ISBN 3-540-44179-4. (cited on p. 45)

[47] Abdulmotaleb El-Saddik, Abu Saleh Md. Mahfujur Rahman, Souhail Abdala, and Bogdan Solomon. PECOLE: P2P multimedia collaborative environment. *Multimedia Tools Appl.*, 39(3):353–377, 2008. (cited on p. 85)

[48] Facebook. Introducing graph search, 2013. `https://www.facebook.com/about/graphsearch`. (cited on p. 68)

[49] Antonino Famulari and Artur Hecker. Mantle: A novel dosn leveraging free storage and local software. In Vincent Guyot, editor, *ICAIT*, volume 7593 of *Lecture Notes in Computer Science*, pages 213–224. Springer, 2012. ISBN 978-3-642-38226-0. (cited on p. 86)

[50] Faroo. P2P search, 2013. `http://www.faroo.com/hp/p2p/p2p.html`. (cited on p. 68)

[51] Stephen Farrell and Hannes Tschofenig. RFC 7258 – pervasive monitoring is an attack, May 2014. URL `https://tools.ietf.org/html/rfc7258`. (cited on p. 100)

[52] Nick Feamster and Roger Dingledine. Location diversity in anonymity networks. In *WPES*. ACM, 2004. URL `https://nymity.ch/tor-dns/pdf/Feamster2004a.pdf`. (cited on pp. 104 and 105)

[53] David Fifield. meek-google suspended for terms of service violations (how to set up your own), June 2016. URL `https://lists.torproject.org/pipermail/tor-talk/2016-June/041699.html`. (cited on p. 129)

[54] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. *PoPETS*, 2015(2), 2015. URL `https://nymity.ch/tor-dns/pdf/Fifield2015a.pdf`. (cited on p. 129)

[55] Philip W. L. Fong, Mohd M. Anwar, and Zhen Zhao. A privacy preservation model for facebook-style social network systems. In Michael Backes and Peng Ning, editors, *ESORICS*, volume 5789 of *LNCS*, pages 303–320. Springer, 2009. ISBN 978-3-642-04443-4. (cited on p. 68)

[56] Warwick Ford and B.S. Kaliski Jr. Server-assisted generation of a strong secret from a password. In *WET ICE*, pages 176–180. IEEE Computer Society, 2000. ISBN 0-7695-0798-0. (cited on pp. 49 and 60)

[57] Miguel Freitas. Twister – a P2P microblogging platform. *CoRR*, Dec 2013. URL `http://arxiv.org/abs/1312.7152`. (cited on pp. 15, 17, and 86)

[58] Niklas Frykholm and Ari Juels. Error-tolerant password recovery. In *CCS*, pages 1–9. ACM, 2001. ISBN 1-58113-385-5. (cited on pp. 46, 55, and 65)

[59] Lixin Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions on Networking*, 9(6), 2001. URL `https://nymity.ch/tor-dns/pdf/Gao2001a.pdf`. (cited on p. 104)

[60] Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. *IACR Cryptology ePrint Archive*, 2013:622, 2013. (cited on p. 16)

[61] Henri Gilbert and Helena Handschuh. Security analysis of SHA-256 and sisters. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 175–193. Springer, 2003. ISBN 3-540-21370-8. (cited on p. 90)

[62] Benjamin Greschbach, Gunnar Kreitz, and Sonja Buchegger. The devil is in the metadata - new privacy challenges in decentralised online social networks. In *PerCom Workshops*, pages 333–339. IEEE, 2012. ISBN 978-1-4673-0905-9. (cited on p. 84)

[63] Benjamin Greschbach, Gunnar Kreitz, and Sonja Buchegger. User search with knowledge thresholds in decentralized online social networks. In Marit Hansen, Jaap-Henk Hoepman, Ronald E. Leenes, and Diane Whitehouse, editors, *Privacy and Identity Management*, volume 421 of *IFIP Advances in ICT*, pages 188–202. Springer, 2013. ISBN 978-3-642-55136-9. (cited on p. 86)

[64] Ralph Gross and Alessandro Acquisti. Information revelation and privacy in online social networks. In *WPES*, pages 71–80. ACM, 2005. (cited on pp. 5 and 31)

[65] Seda Gürses and al. SPION D2.1 - State of the Art. In Seda Gürses, editor, *SBO Security and Privacy for Online Social Networks*. 2011. (cited on pp. 1 and 30)

[66] Seda Gurses and Claudia Diaz. Two tales of privacy in online social networks. *IEEE Security & Privacy*, 11(3):29–37, 2013. ISSN 1540-7993. (cited on p. 6)

[67] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security*. USENIX, 2016. URL `https://nymity.ch/tor-dns/pdf/Hayes2016a.pdf`. (cited on pp. 105 and 117)

[68] Cormac Herley and Paul C. van Oorschot. A research agenda acknowledging the persistence of passwords. *IEEE Security & Privacy*, 10(1):28–36, 2012. (cited on p. 44)

[69] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *CCSW*. ACM, 2009. URL `https://nymity.ch/tor-dns/pdf/Herrmann2009a.pdf`. (cited on p. 105)

[70] International Electrotechnical Commission. ISO/IEC 27002:2005. Information technology – Security techniques – Code of practice for information security management, 2005. (cited on pp. 48 and 54)

[71] Tomas Isdal, Michael Piatek, Arvind Krishnamurthy, and Thomas E. Anderson. Privacy-preserving P2P data sharing with OneSwarm. In *SIG COMM*, pages 111–122. ACM, 2010. ISBN 978-1-4503-0201-2. (cited on pp. 44 and 45)

[72] Rob Jansen and Aaron Johnson. Safely measuring Tor. In *CCS*. ACM, 2016. URL `https://nymity.ch/tor-dns/pdf/Jansen2016a.pdf`. (cited on p. 113)

[73] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Computing Surveys*, 25(3), 2007. (cited on p. 46)

[74] Raúl Jiménez, Flutra Osmani, and Björn Knutsson. Sub-second lookups on a large-scale Kademlia-based overlay. In *P2P*, pages 82–91. IEEE Computer Society, 2011. ISBN 978-1-4577-0150-4. (cited on pp. 46, 62, 63, and 65)

[75] Aaron Johnson. The Tor path simulator. URL `https://github.com/torps/torps`. (cited on pp. 101, 104, and 123)

[76] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on Tor by realistic adversaries. In *CCS*. ACM, 2013. URL `https://nymity.ch/tor-dns/pdf/Johnson2013a.pdf`. (cited on pp. 100, 104, 105, 124, and 126)

[77] Peter Johnson-Lenz and Trudy Johnson-Lenz. Groupware: Coining and defining it. *SIGGROUP Bull.*, 19(2):34–, August 1998. (cited on p. 85)

[78] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *CCS*. ACM, 2014. URL `https://nymity.ch/tor-dns/pdf/Juarez2014a.pdf`. (cited on pp. 105 and 117)

[79] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. Toward an efficient website fingerprinting defense. In *ESORICS*. Springer, 2016. URL `https://nymity.ch/tor-dns/pdf/Juarez2016a.pdf`. (cited on pp. 103, 106, and 119)

[80] Joshua Juen, Aaron Johnson, Anupam Das, Nikita Borisov, and Matthew Caesar. Defending Tor from network adversaries: A case study of network path prediction. *PoPETS*, 2015(2), 2015. URL `https://nymity.ch/tor-dns/pdf/Juen2015a.pdf`. (cited on pp. 104, 105, 124, and 125)

[81] Sheharbano Khattak, David Fifield, Sadia Afroz, Mobin Javed, Srikanth Sundaresan, Damon McCoy, Vern Paxson, and Steven J. Murdoch. Do you see what I see? differential treatment of anonymous users. In *NDSS*. The Internet Society, 2016. URL `https://nymity.ch/tor-dns/pdf/Khattak2016a.pdf`. (cited on pp. 114 and 115)

[82] Min Kyung Kim and Hee-Cheol Kim. Awareness and privacy in groupware systems. In *CSCWD*, pages 984–988. IEEE, 2006. ISBN 1-4244-0165-8. (cited on p. 85)

[83] Kate Krauss. Ethical Tor research: Guidelines, November 2015. URL `https://blog.torproject.org/blog/ethical-tor-research-guidelines`. (cited on p. 129)

[84] Balachander Krishnamurthy and Craig E. Wills. On the leakage of personally identifiable information via online social networks. *SIGCOMM Comput. Commun. Rev.*, 40:112–117, 2010. ISSN 0146-4833. (cited on pp. 6 and 31)

[85] Amanda Lenhart and Mary Madden. Social networking websites and teens: An overview. *Pew Internet project data memo*, 13, 2007. (cited on p. 32)

[86] Jinyang Li, Boon Thau Loo, Joseph M. Hellerstein, M. Frans Kaashoek, David R. Karger, and Robert Morris. On the feasibility of peer-to-peer web indexing and search. In M. Frans Kaashoek and Ion Stoica, editors, *IPTPS*, volume 2735 of *LNCS*, pages 207–215. Springer, 2003. ISBN 3-540-40724-3. (cited on p. 68)

[87] W. D. Li, Soh-Khim Ong, Jerry Y. H. Fuh, Yoke San Wong, Y. Q. Lu, and Andrew Y. C. Nee. Feature-based design in a distributed and collaborative environment. *Computer-Aided Design*, 36(9):775–797, 2004. (cited on p. 85)

[88] Mark Lillibridge, Sameh Elnikety, Andrew Birrell, Michael Burrows, and Michael Isard. A cooperative internet backup scheme. In *USENIX*, pages 29–41. USENIX, 2003. ISBN 1-931971-10-2. (cited on p. 45)

[89] Ingrid Lunden. Facebook turns off facial recognition in the EU, gets the all-clear on several points from Ireland's data protection commissioner on its review. Online, Sep 2012. URL `http://techcrunch.com/2012/09/21/facebook-turns-off-facial-recognition-in-the-eu-gets-the-all-clear/`. (cited on p. 84)

[90] Aniket Mahanti, Niklas Carlsson, Anirban Mahanti, Martin Arlitt, and Carey Williamson. A tale of the tails: Power-laws in Internet measurements. *IEEE Network*, 27(1), 2013. URL `https://nymity.ch/tor-dns/pdf/Mahanti2013a.pdf`. (cited on p. 111)

[91] Nick Mathewson. Remove global client-side DNS caching, July 2012. URL `https://gitweb.torproject.org/torspec.git/tree/proposals/205-local-dnscache.txt`. (cited on p. 104)

[92] G. Mega, A. Montresor, and G. P. Picco. *Efficient dissemination in decentralized social networks*, page 338–347. Aug 2011. (cited on p. 14)

[93] Mixmaster. URL `http://mixmaster.sourceforge.net`. (cited on p. 100)

[94] Steven J. Murdoch and Piotr Zieliński. Sampled traffic analysis by Internet-exchange-level adversaries. In *PET*. Springer, 2007. URL `https://nymity.ch/tor-dns/pdf/Murdoch2007a.pdf`. (cited on pp. 100, 104, 106, and 110)

[95] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Oct 2008. URL `http://www.bitcoin.org/bitcoin.pdf`. (cited on pp. 15 and 87)

[96] M. A. U. Nasir, S. Girdzijauskas, and N. Kourtellis. *Socially-aware distributed hash tables for decentralized online social networks*, page 1–10. Sep 2015. (cited on p. 12)

[97] Netcraft. July 2016 web server survey, July 2016. URL `https://news.netcraft.com/archives/2016/07/19/july-2016-web-server-survey.html`. (cited on pp. 111 and 122)

[98] Helen Nissenbaum. Privacy as Contextual Integrity. *Washington Law Review*, 79:119, 2004. (cited on p. 5)

[99] Rishab Nithyanand, Oleksii Starov, Adva Zair, Phillipa Gill, and Michael Schapira. Measuring and mitigating AS-level adversaries against Tor. In *NDSS*. The Internet Society, 2016. URL `https://nymity.ch/tor-dns/pdf/Nithyanand2016a.pdf`. (cited on p. 104)

[100] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. Website fingerprinting at Internet scale. In *NDSS*. The Internet Society, 2016. URL `https://nymity.ch/tor-dns/pdf/Panchenko2016a.pdf`. (cited on pp. 105 and 110)

[101] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networls. In *WPES*. ACM, 2011. URL `https://nymity.ch/tor-dns/pdf/Panchenko2011a.pdf`. (cited on p. 105)

[102] Thomas Paul, Sonja Buchegger, and Thorsten Strufe. Decentralized social networking services. In Luca Salgarelli, Giuseppe Bianchi, and Nicola Blefari-Melazzi, editors, *Trustworthy Internet*, chapter 14, pages 187–199. Springer, 2011. (cited on pp. 5, 9, and 31)

[103] Thomas Paul, Antonino Famulari, and Thorsten Strufe. A survey on decentralized online social networks. *Computer Networks*, 75, Part A:437–452, Dec 2014. ISSN 1389-1286. (cited on pp. 9 and 11)

[104] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77, Dec 2007. ISSN 0742-1222. (cited on p. 20)

[105] Mike Perry. Padding negotiation, September 2015. URL `https://gitweb.torproject.org/torspec.git/tree/proposals/254-padding-negotiation.txt`. (cited on pp. 103, 106, and 119)

[106] Bogdan C. Popescu, Bruno Crispo, and Andrew S. Tanenbaum. *Safe and Private Data Sharing with Turtle: Friends Team-Up and Beat the System*. Springer, 2004. URL `http://www.cs.vu.nl/~ast/afscheid/publications/sec_prot-2004.pdf`. (cited on p. 17)

[107] Niels Provos and David Mazières. A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track*, pages 81–91. USENIX, 1999. ISBN 1-880446-32-4. (cited on pp. 60 and 64)

[108] Ariel Rabkin. Personal knowledge questions for fallback authentication: security questions in the era of Facebook. In *SOUPS*, pages 13–23. ACM, 2008. ISBN 978-1-60558-276-4. (cited on p. 55)

[109] Walter Reinhard, Jean Schweitzer, Gerd Völksen, and Michael Weber. CSCW tools: Concepts and architectures. *IEEE Computer*, 27(5):28–36, 1994. (cited on p. 85)

[110] Sean C. Rhea, Patrick R. Eaton, Dennis Geels, Hakim Weatherspoon, Ben Y. Zhao, and John Kubiatowicz. Pond: The OceanStore prototype. In *FAST*. USENIX, 2003. (cited on p. 46)

[111] RIPE Network Coordination Centre. RIPE Atlas. URL `https://atlas.ripe.net/`. (cited on pp. 102, 105, and 125)

[112] Tom Rodden and Gordon S. Blair. CSCW and distributed systems: The problem of control. In Liam J. Bannon, Mike Robinson, and Kjeld Schmidt, editors, *ECSCW*. Kluwer, 1991. ISBN 0-7923-1439-5. (cited on p. 85)

[113] Michael Rogers and Saleem Bhatti. *Private Peer-to-Peer Networks*. Springer US, 2010. ISBN 978-0-387-09750-3. URL `http://link.springer.com/10.1007/978-0-387-09751-0`. (cited on p. 12)

[114] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Rachid Guerraoui, editor, *Middleware*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer, 2001. ISBN 3-540-42800-3. (cited on p. 86)

[115] Avi Rushinek and Sara F. Rushinek. What makes users happy? *Commun. ACM*, 29(7):594–598, 1986. (cited on pp. 62 and 65)

[116] Krzysztof Rzadca, Anwitaman Datta, and Sonja Buchegger. Replica placement in P2P storage: Complexity and game theoretic analyses. In *ICDCS*, pages 599–609. IEEE Computer Society, 2010. ISBN 978-0-7695-4059-7. (cited on pp. 62 and 64)

[117] Andrea De Salve, Paolo Mori, and Laura Ricci. *A Privacy-Aware Framework for Decentralized Online Social Networks*, page 479–490. Lecture Notes in Computer Science. Springer International Publishing, Sep 2015. ISBN 978-3-319-22851-8. (cited on p. 12)

[118] Malte Schwarzkopf, Anil Madhavapeddy, Theodore Hong, and Richard Mortier. Personal Containers: Yurts for Digital Nomads. *perscon.net*, 2011. URL `http://perscon.net/papers/digital-yurts-draft1.pdf`. (cited on p. 9)

[119] Garry Scoville. Good security questions. `http://goodsecurityquestions.com/` (19th April, 2012). (cited on p. 55)

[120] R. Sharma and A. Datta. Supernova: Super-peers based architecture for decentralized online social networks. In *2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012)*, Jan 2012. (cited on pp. 12 and 31)

[121] Gerry Shih. Facebook admits year-long data breach exposed 6 million users. Online, Jun 2013. URL `http://www.reuters.com/article/2013/06/21/net-us-facebook-security-idUSBRE95K18Y20130621`. (cited on p. 84)

[122] Cooper Smith. Reinventing social media: Deep learning, predictive marketing, and image recognition will change everything, Mar 2014. URL http://www.businessinsider.com/social-medias-big-data-future-2014-3. (cited on p. 84)

[123] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. RAPTOR: Routing attacks on privacy in Tor. In *USENIX Security*. USENIX, 2015. URL https://nymity.ch/tor-dns/pdf/Sun2015a.pdf. (cited on p. 105)

[124] Team Cymru. IP to ASN mapping. URL https://www.team-cymru.org/IP-ASN-mapping.html. (cited on p. 107)

[125] The Tor Project. CollecTor. URL https://collector.torproject.org. (cited on pp. 113 and 124)

[126] The Tor Project. Tor Metrics–Top-10 countries by directly connecting users. URL https://metrics.torproject.org/userstats-relay-table.html. (cited on p. 124)

[127] Niraj Tolia, David G. Andersen, and Mahadev Satyanarayanan. Quantifying interactive user experience on thin clients. *IEEE Computer Society*, 39(3): 46–52, 2006. (cited on pp. 64 and 65)

[128] Jonathan Trevor, Thomas Koch, and Gerd Woetzel. Metaweb: Bringing synchronous groupware to the world wide web. In *ECSCW*, pages 65–80, 1997. (cited on p. 85)

[129] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. A survey of DHT security techniques. *ACM Computing Surveys*, 43(2):8, 2011. (cited on pp. 60 and 61)

[130] Population Division U.S. Census Bureau. Genealogy data: Frequently occurring surnames from census 1990, 1990. URL http://www.census.gov/genealogy/www/data/1990surnames/names_files.html. (cited on p. 77)

[131] Population Division U.S. Census Bureau. Table 1. annual estimates of the resident population for incorporated places over 50,000, ranked by july 1, 2011 population: April 1, 2010 to july 1, 2011 (sub-est2011-01), 2012. URL http://www.census.gov/popest/data/cities/totals/2011/tables/SUB-EST2011-01.csv. (cited on p. 77)

[132] Le-Hung Vu, Karl Aberer, Sonja Buchegger, and Anwitaman Datta. Enabling secure secret sharing in distributed online social networks. In *ACSAC*, pages 419–428. IEEE Computer Society, 2009. ISBN 978-0-7695-3919-5. (cited on pp. 46 and 64)

[133] L. Wang and J. Kangasharju. *Real-world sybil attacks in BitTorrent mainline DHT*, page 826–832. Dec 2012. (cited on p. 12)

[134] Tao Wang. *Website Fingerprinting: Attacks and Defenses*. PhD thesis, University of Waterloo, 2015. URL `https://nymity.ch/tor-dns/pdf/Wang2015a.pdf`. (cited on pp. 103 and 105)

[135] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *USENIX Security*. USENIX, 2014. URL `https://nymity.ch/tor-dns/pdf/Wang2014a.pdf`. (cited on pp. 103, 105, and 116)

[136] Tao Wang and Ian Goldberg. Improved website fingerprinting on Tor. In *WPES*. ACM, 2013. URL `https://nymity.ch/tor-dns/pdf/Wang2013a.pdf`. (cited on pp. 105 and 117)

[137] Tao Wang and Ian Goldberg. On realistically attacking Tor with website fingerprinting. *PoPETs*, 2016(4), 2016. URL `https://nymity.ch/tor-dns/pdf/Wang2016a.pdf`. (cited on p. 105)

[138] Samuel D. Warren and Louis D. Brandeis. The Right to Privacy. *Harvard Law Review*, 4(5):193–220, December 1890. (cited on p. 5)

[139] Klaus Wehrle, Stefan Götz, and Simon Rieche. Distributed hash tables. In *Peer-to-Peer Systems and Applications*, volume 3485, pages 79–93. Springer, 2005. ISBN 3-540-29192-X. (cited on p. 46)

[140] Alan F. Westin. *Privacy and Freedom*. New York : Atheneum for the Assoc. of the Bar of the City of New York, 1967. ISBN 9780370013251. (cited on p. 5)

[141] Philipp Winter. exitmap: A fast and modular scanner for Tor exit relays. URL `https://github.com/NullHypothesis/exitmap`. (cited on p. 108)

[142] Philipp Winter, Richard Köwer, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog, and Edgar Weippl. Spoiled onions: Exposing malicious Tor exit relays. In *PETS*. Springer, 2014. URL `https://nymity.ch/tor-dns/pdf/Winter2014b.pdf`. (cited on pp. 108 and 129)

[143] David L. Word, Charles D. Coleman, Robert Nunziata, and Robert Kominski. Demographic aspects of surnames from census 2000, 2000. URL `http://www.census.gov/genealogy/www/surnames.pdf`. (cited on p. 77)

[144] Ching-man Au Yeung, Ilaria Liccardi, Kanghao Lu, Oshani Seneviratne, and Tim Berners-Lee. Decentralization: The future of online social networking. In *W3C Workshop on the Future of Social Networking Position Papers*, volume 2, page 2–7, 2009. (cited on p. 9)

[145] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. *IEEE/ACM Trans. Netw.*, 18(3):885–898, Jun 2010. ISSN 1063-6692. (cited on p. 12)

[146] Guozhen Zhang and Qun Jin. Scalable information sharing utilizing decentralized p2p networking integrated with centralized personal and group media tools. In *AINA (2)*, pages 707–711. IEEE Computer Society, 2006. ISBN 0-7695-2466-4. (cited on p. 85)

[147] Liang Zhu, Zi Hu, John Heidemann, Duane Wessels, Allison Mankin, and Nikita Somaiya. Connection-oriented DNS to improve privacy and security. In *Security & Privacy*. IEEE, 2015. URL `https://nymity.ch/tor-dns/pdf/Zhu2015a.pdf`. (cited on p. 130)

[148] Mary Ellen Zurko. *IBM Lotus Notes/Domino: Embedding Security in Collaborative Applications*, chapter 30, pages 607–622. O'Reilly Media, Inc., 2005. ISBN 0596008279. (cited on p. 85)