UNIVERSITY OF CAMBRIDGE

Securing encrypted communication

Diana-Alexandra Vasile

St Edmund's College

This dissertation is submitted on 7th July 2023 for the degree of Doctor of Philosophy

DECLARATION

Diana-Alexandra Vasile
July, 2023

Abstract

Securing encrypted communication

Diana-Alexandra Vasile

Secure messaging has led to the mass adoption of strong security principles such as end-to-end encryption and perfect forward secrecy, which had previously failed to gain traction. While this marks tremendous progress in the effort to enhance the security of communication, there are still many open challenges. This dissertation looks at two open problems: providing key transparency in secure messaging apps in an attempt to detect targeted wiretaps, and securing the initial contact for journalists.

We begin by formalising the different combinations of key-to-name bindings seen in popular secure messaging apps into key-name-graphs, which we then use to verify that the key server provides the same snapshot of a key-name-graph for a user to all his friends. This approach is proposed as a baseline gossip protocol between friends who physically co-locate, however, when coupled with some enhancements, it has broader applicability to both different underlying network technologies and to expanding verification beyond the friendship connection. We analyse the deployability of the baseline gossip protocol using secondary data from two datasets: Wi-Fi usage and call-detail records. We also implement the protocol as a discrete event simulator and use it to perform model checking to analyse the protocol's security.

Secure messaging is not enough for everyone, though. There are certain cases in which further enhancements such as anonymity and metadata privacy are needed to protect those communicating. As such, we analysed the options available to journalists to communicate with sources who may later become whistleblowers. Through the insights from two workshops organised with large British news organisations we show that the options available to journalists are inadequate. We identify several open problems, such as low-latency secure and anonymous communication and secure collaboration. We focus our efforts on initial contact, a problem that appeared during the workshop to have a significant detrimental effect on the security of sources. We discovered that often sources do not place significant emphasis on secure communication from the start, and retrospectively applying security is non-trivial by the time they are ready to share sensitive information. We thus propose a new and secure initial contact system, called CoverDrop, which is integrated as a secure library directly inside newsreader apps.

ACKNOWLEDGEMENTS

Eight years in the making. It has truly been an adventure: a family created, two children born and raised, three house moves and a global pandemic. But the light at the end of the tunnel is finally within reach and there are many people I would like name to thank for contributing to the research, supporting me personally, and anything in between.

First, I would like to thank my supervisor, Professor Alastair Beresford, without whose guidance, support and understanding I would not have made it until the end of this journey and without whom I wouldn't have grown into a fully-fledged researcher. Thank you for teaching me what research is, pushing me to explore the boundary, giving me confidence and reminding me that it's okay to have a different path than my peers.

Our Digital Technology Group (DTG) is always an amazing environment, filled with interesting discussions, research ideas and experiences. It has felt like home for so many years, even through all my interruptions it was always so good to be back! I would like to thank Daniel Thomas, Martin Kleppmann, Stan Zhang, Jovan Powar, Daniel Hugenroth, Mansoor Ahmed-Rengers, Michael Dodson, Stephan Kollmann, Ceren Kocaogullar, and Luis Adan Saavedra del Toro for your friendship, support and collaboration. Within the wider lab, I would like to thank Ross Anderson, Alice Hutchings, Andy Rice, and Andy Hopper for being so inspirational and sharing their experience with me.

Outside the lab, I am grateful to Dr. Clare Oliver-Williams for her generous support in providing play dates for my children and to Jeunese Adrienne Payne for her invaluable feedback and proofreading. I also want to thank my close friends, Luana and Flaviu Bulat, Iustina and Matt Woodacre, Daniela Radu, and Bogdan Spiridon, for their unwavering moral support.

I would also like to thank Nokia Bell Labs, in particular Prof Fahim Kawsar, who not only funded my research, but also gave me and my research a new home.

I couldn't go without saying thanks to my family and my parents for their support, continuous worry and love. Of all, the one who touched this work most is my aunt, Anca Crisan, who not only is an amazing life coach, but also proof read the work many items over these past few years and provided invaluable feedback on writing style.

Finally, to my husband, Catalin, without whom this PhD wouldn't have happened. You believed in me and encouraged me to try for a masters, then a PhD and when times were hard you were always there – you and our boys, Mircea and Matei! I thank and love you!

CONTENTS

CHAPTER 1

INTRODUCTION

Decades of work went into securing communication, from enhanced security for devices, to securing network traffic and applications. Significant effort has been placed on ensuring both message content privacy, through which the actual content of communication is protected, and metadata privacy, through which additional information about the communication, such as the sender, recipient, timestamp and location data, is protected. The deployment of end-to-end encryption to secure messaging apps, which is now widely adopted by more than 2 billion active users, is a robust mechanism to protect message content. This ensures that communication remains encrypted throughout the transmission process. Also, techniques such as onion-routing-based messaging and file transfer have been deployed to address the privacy concerns posed by not protecting metadata. This approach ensures that important metadata, such as the sender's identity and the intended recipient, is obscured by the time the message is delivered or sent.

The Snowden revelations in 2013 validated the risk posed by insecure communication on backend network infrastructure between datacenters, and demonstrated to the wider public that mass surveillance of civilians and industrial espionage is real. As a direct result, we saw a wide-spread adoption of Transport Layer Security (TLS), both between browsers and servers, but also between backend services. Furthermore, public demand for higher protection of personal data increased and many organisations have now adopted stronger data security measures, including end-to-end encryption, multi-factor authentication, and secure data storage practices. These measures help to safeguard personal data against unauthorised access, ensuring that user information remains secure and private.

In most cases, using TLS to encrypt the data exchange between devices and servers is sufficient. However, devices are increasingly communicating directly with other end devices with the server temporarily storing and then forwarding the communication to the final recipient, which increases the risk of potential eavesdropping and interception of sensitive information. This is because although TLS provides client-to-server security through encryption, in order to forward the message to the intended recipient, the server receives a message encrypted for itself, it decrypts it, checks the recipient and re-encrypts the message with the correct key for the recipient before forwarding it. This process leaves the communication vulnerable to attacks, making this high concentration of information at the server an attractive target to malicious actors.

End-to-end encryption enhances protection for personal data – data is encrypted at the sending device and only decrypted at the receiving device. This takes away the trust placed in the infrastructure to keep personal data safe. While end-to-end encrypted solutions such as PGP (encrypted emails) and OTR (encrypted messaging) existed for over two decades, they never raised in popularity particularly due to usability and scalability problems. Secure messaging, though, is by far the biggest end-to-end encrypted deployment, with popular apps such as iMessage and WhatsApp being used by billions of active users daily [13, 226]. Other areas, such as different types of collaboration software, are working on usable end-to-end encrypted solutions [116, 122].

Current encryption standards do provide stronger protection for personal data. Bug bounty programs, responsible disclosure approaches and zero-day vulnerabilities are all meant to help software vendors improve the security of their software and minimise risk of attack. Key ratcheting, with forward secrecy and future secrecy properties, now ensure that encryption keys are short-lived: they are discarded when the message is encrypted and rolled over to new keys such that an attack at the current time will only affect the messages encrypted with that key, and will not give the attacker a view of previous or any future messages. Since breaking the encryption is becoming harder, requests for exceptional access by law enforcement and political leaders are increasing by claiming it impedes investigation into criminal activity [2, 131]. Such proposals include key escrow systems, or *authorised* backdoors into the encryption protocol. Another approach recently proposed by GCHQ to wiretap end-to-end encrypted messages without having to modify any of the encryption, or rely on backdoors, by silently introducing an extra end point targeted to a certain user's account, which will allow access to any messages the user sends or receives [131]. While the former options are harder to perform at scale and more expensive in terms of compute time, the method proposed by GCHQ is a subtle, quick and inexpensive approach.

Operator accountability thus becomes as important as the demand for user data protection, since trust has now moved from the messaging architecture to the trust establishment architecture [213]. In practice, Signal and WhatsApp provide this through manual verification to allow users to manually compare a short alphanumeric string, or for one user to take a picture with their smartphone of a QR-code displayed on the other user's smartphone. This step is optional since both services run a central public key infrastructure (PKI).

Previous automated solutions attempt to decentralise power by displacing and reducing the trust placed in certificate authorities [203], or by encouraging certificate authorities to submit the contents of their PKI to public append-only logs in order to support external audits [127, 146].

None of these solutions can guarantee correctness, but they attempt to detect misbehaviour after-the-fact. The main challenges with these solutions include interoperability between the operators of end-to-end encrypted services (closed vs. federated), the requirement for operators to cooperate and share the contents of their key directories, and the requirement that users perform manual key checking.

Furthermore, this end-to-end encrypted approach does not protect metadata by default. Despite not revealing the message contents, it still reveals information about the participants to a network observer, such as who is communicating to whom and when, location, and IP addresses. Mining such information reveals communication or travel patterns and other vital information that can be used by attackers. To protect the metadata of communication we can run these applications over anonymity systems such as Tor, Mixminion, or Loopix, which provide a higher level of metadata privacy and censorship resistance, but it is not straightforward to apply such a solution in all cases. Also, aside from mining information at population scale, metadata can also reveal such information useful in a targeted attack. Since in some situations we are presented with increased risk for individuals, just running an anonymity system such as Tor can single out a target. For instance, whistleblowers have faced the consequences in the recent past, such as dismissal [172], imprisonment [208, 230], intimidation [153], and even assassination [207].

Lastly, striking a good balance between usability (driving adoption rates) and security is difficult. There are famous examples of systems that were designed for increased security but suffered from usability issues, leading to low adoption rates [182, 192, 228]. On the other end of the spectrum are popular apps such as Signal, WhatsApp and iMessage, which win on usability by automating most of the security requirements and running their own PKI as an online service, thus enabling a seamless experience for their users.

The driver behind this research is that everyone deserves good security and protection of their personal data. This need was augmented recently with the rise in location independence due to remote working during the SARS-Cov2 global pandemic, which has changed the way people work through the introduction of social distancing restrictions in early 2020 in many parts of the world. Different sectors, such as education, judicial and health have had to adapt to operating online on a daily basis. Furthermore, the fact that remote work has been possible for such an extended period of time makes it probable that hybrid work models will persist moving forward [134]. This stresses the importance of having secure systems to facilitate different communication needs.

1.1 Contributions

In this dissertation we set out to secure communication for end-users by targeting two main areas: detecting targeted wiretaps for popular secure messaging apps, and providing potential whistle-blowers, a subset of end-users with higher needs for secure and anonymous communication, with a system to securely initiate contact with investigative journalists.

In summary, the main contributions of this research are:

- An analysis and subsequent formalisation of public key-to-name bindings into key-name graphs (*KNGs*), which can represent all current combinations of bindings in production end-to-end encrypted apps (Chapter 3).
- A novel gossip protocol for key verification, which combines the benefits of automation and web-of-trust style distributed key checking (Chapter 3).
- An implementation of the gossip protocol steps into a discrete event simulator to analyse the protocol efficacy through model checking on all the types of social ties that can exist with up to 4 users, including a stochastic exploration of the step tree of possible events that can alter a KNG (Chapter 4).
- An analysis of the deployability of the protocol with Wi-Fi usage data from 11 485 Android devices and a social-geographic analysis of over 67 million call-detail records belonging to over 1 million GSM subscribers, over a 35-day period (Chapter 4).
- The system requirements for a secure initial contact mechanism based on actual journalists' experiences, based on which we develop a realistic adversarial model (Chapter 5).
- A secure initial contact mechanism in three components: (1) a secure mobile library; (2) a TEE-based mix strategy; and (3) a networking model that supports integration into existing CDN-based networks with minimal modification (Chapter 6).

All experiments reported in this dissertation followed University of Cambridge's ethical research policy as approved by the Ethics Committee of the University of Cambridge, Department of Computer Science and Technology. There are two main types of research involving humans or their devices for which we considered the ethical implications and received approval from the departmental ethics committee in advance of starting the research: (A) the deployability analysis of our gossip protocol, based on data from two datasets recording information about users' devices (Chapter 4); (B) and the system requirements capture for our secure initial contact mechanism based on actual journalists' experiences, based on data coded from and quotes derived from two workshops that we organised in London in late 2019 (Chapter 5). Any particular ethical considerations and procedures will be discussed in the relevant chapters, where appropriate.

1.2 Publications

Parts of the research described in this dissertation have been published in the proceedings of peer reviewed conferences and workshops:

1. “From Secure Messaging to Secure Collaboration”, Martin Kleppmann, Stephan A. Kollmann, Diana A. Vasile, and Alastair R. Beresford, at Security Protocols Workshop 2018 (SPW’18) [116].

   This is a position paper and is the result of lengthy discussions between all authors and it was written in large by Martin. The work in this paper strenghtens the need for secure collaboration approaches, particularly looking at end-to-end encrypted communications. It thus analyses the required changes in principles and architecture to move the end-to-end encrypted implementations directly from secure messaging to a collaborative system such as Google Docs. Ideas inspired from it are reflected in several places throughout this dissertation, such as Chapters 1, 2, 3 and 5.

2. “Ghost trace on the wire? Using key evidence for informed decisions”, Diana A. Vasile, Daniel R. Thomas and Alastair R. Beresford, at Security Protocols Workshop 2019 (SPW’19) [217].

   This is a position paper analysing the types of key-change events that happen in a secure messaging application. The main contribution of this work is to analyse what kind of cryptographic evidence we can gather prior to this key-change events taking place to help improve the notifications users see. This work is the foundational work for the gossip protocol in Chapter 3 and Chapter 4. Ideas inspired from it are also seen in Chapter 2. The paper is written mostly by myself. Alastair and Daniel contributed with valuable discussions, as well as feedback on writing. Daniel also assisted in thinking about and writing the section on acquiring evidence for key validity.

3. “CoverDrop: Blowing the whistle through a news app”, Mansoor Ahmed-Rengers, Diana A. Vasile, Daniel Hugenroth, Alastair R. Beresford and Ross Anderson, in submission to Privacy and Enhancing Technologies Symposium (PETS’2022).

   This paper proposes a new system for sources to initiate contact with journalists in a secure way and forms the basis of Chapters 5 and 6. The idea to investigate and create a system to secure communication between journalists and their sources originated from Alastair and was initially shaped through lengthy discussions between myself and Alastair. It was then enriched by Mansoor, Daniel and Ross joining the team. The overall system idea emerged from lengthy discussions between all the authors. I led the organisation, requirements capture and coding of the workshops in London, as well as performed the analysis of current systems in use by journalists. I also performed most of the security analysis, with important contributions from Daniel, particularly in the unobservability of communication and plausible deniability when device confiscation occurs. Daniel coded about 90% of the prototype implementation and did the performance evaluation. Mansoor coded the rest of 10% of the prototype implementation and focused on writing most of the paper, with

Daniel covering the performance evaluation and app prototypes and myself the status quo and the security analysis.

CHAPTER 2

BACKGROUND

We provide a brief history of encryption, starting from rudimentary attempts at encryption and finishing with modern encrypted communication. This illustrates some of the ongoing concerns, open problems, and approaches in encrypted communication, which we build on in subsequent chapters, such as trust establishment, transparency, and usable security. We also outline an overview of different approaches to providing secure communication as well as to providing anonymity.

2.1 Encrypted communication

The desire and need to keep messages secret is prevalent throughout history, however this was not always acknowledged or even possible. We provide a brief outlook on early encryption and show the transition to modern encryption. At the base of working encryption lies trust establishment which we cover in Section 2.1.2, because without a means to exchange initial information securely one cannot have encrypted communication between two or more parties.

2.1.1 Early encryption – A brief history

A wide variety of examples of early encryption exist, from manual encoding, such as unknown hieroglyphs on Egyptian tombs, or the use of the scytale transposition cipher [111], to more complex examples involving mechanical encryption devices, such as the Enigma machine used during World War II [88]. This all paved the way to present day efforts, which see end-to-end encryption become the default communication option in popular messaging apps. We briefly review in this section some of the significant milestones that were achieved to make present day encryption possible.

Before the 1970s, encryption was typically used by governments and research was often classified. The early 70s mark a significant shift with the proposal and adoption of the Data

Encryption Standard (*DES*) [49], as well as the invention of public-key cryptography, which was shown to be theoretically possible in the work of Whitfield Diffie and Martin Hellman on the Diffie-Hellman key exchange [56]. A year later, the Rivest-Shamir-Adleman encryption scheme (*RSA*) [178] was the first public-private key-based encryption scheme to be implemented. Over time, DES became the de-facto standard for encryption in the industry, but was later shown vulnerable. DES is most known for its vulnerability to brute-force attacks due to the small 56-bit key size[1], which was shown in 1999 to be breakable in less than 24 hours on a personal computer [197]. The Advanced Encryption Standard (AES) was introduced in the early 2000s to address some of DES's known vulnerabilities [64]. The most significant difference to DES is the algorithm selection process used for AES, which was open to the public to submit their own algorithms [46]. AES includes larger key sizes to address the fallacies of DES.

With the Internet boom, the need for transport layer security became more evident. The Secure Sockets Layer (*SSL*) protocol was initially proposed by Netscape in 1994; but the first version had weaknesses so was never released. In subsequent years, SSL 2.0 and SSL 3.0 [72, 117] were released and used for over a decade until deprecated due to major deficiencies, such as unprotected handshake messages and the ability for a person-in-the-middle to terminate sessions by inserting a TCP FIN request [170]. Several versions of the Transport Layer Security (*TLS*) protocol were also developed, which interoperated with SSL until it fully replaced it. TLS is now de-facto for modern day encryption.

2.1.2 Modern encryption and trust establishment

After the Internet Boom, there have been two main approaches to enable modern day encryption of data: *asymmetric key encryption* (also known as *public-key encryption*) and *symmetric key encryption*. In symmetric key encryption the communicating parties arrange for a setup phase in which they agree on a shared secret key. This key is then used for both the encryption and decryption of messages. On the other hand, in public-key encryption, each party independently generates a public-private key pair, stores the private key securely and makes their public key known to communicating partners. The private key is used for encryption (or signing) and the public key is used for the decryption (or verification). Both symmetric and asymmetric encryption methods encounter key management problems: the former encryption method has the initial hurdle of users agreeing on a shared secret securely; while the latter problem is slightly different in nature – who holds the public keys and how does one distribute and authenticate their public key? This problem is referred to as the trust establishment problem. We discuss the two main approaches that have been used to address this trust establishment problem: the use of a Public-Key Infrastructure (*PKI*) and the Web-of-Trust (*WoT*) model.

[1]Said to have been insisted on by the NSA during IBM's work on the proposal

Public-Key Infrastructure. One approach to addressing trust establishment is to use a PKI, which is a widely used solution nowadays. PKI provides a mechanism to record unique digital identities for users, devices, and applications and certify the binding between a public key and a unique digital identity, hereafter *name*, by issuing a signed certificate. However, this approach relies on an authority to issue the certificate and validate it. Should access to the private key be lost, the authority is also enabled to revoke that binding and issue another. The most widely used form of PKI is the Certificate Authority (*CA*) used by web browsers when fetching web pages. CAs map DNS names to public keys to help automatically set up HTTPS connections without any manual verification.

Web-of-Trust. An alternative to the centralised PKI trust model is WoT [114], which proposes fully distributed key management. WoT relies on the idea that there will naturally exist multiple "webs of trust" that can be accessed by a user through their friends (*introducers*). Thus, in WoT users manage their own public key to name bindings and certify to the validity of their friends' bindings by *digitally signing* them. Key bindings can receive multiple certificates (i.e. can be signed by more than one friend). Key revocation is a manual and complicated process, particularly because it has to be thought in advance: in case of loss of access to a private key, users cannot revoke the public key unless a revocation message has been prepared prior to the loss of access. Without revocation, loss of access to a private key means that the users could no longer decrypt messages intended for them. Pretty Good Privacy (PGP) [1, 234] was the first to make encryption work for end-users using the WoT approach. With PGP, users provide self-signed certificates and third-party attestations of those certificates. Users manually generate public-private PGP key pairs, store a self-signed version of the public key-to-name binding in independently maintained dedicated servers and then ask friends to attest to binding validity. PGP encountered usability issues which affected its adoption, which we discuss in Section 2.3.

2.2 The need for transparency

There is an inherent trust in authorities to honestly run and distribute the components of the services we use. Users trust Certificate Authorities to issue the correct certificates for valid web services, key servers to distribute the correct key bindings for other users, and package distributors, such as package managers or app stores, to distribute the correct binary for an application they wish to install. Providing transparency in secure systems increases user trust by enabling users to verify that these services are operating honestly rather than trusting them blindly. We discuss in this section three approaches taken to increase transparency and aid verification.

2.2.1 Certificate Transparency

CAs enable trust establishment for DNS entries by issuing a certificate binding a public key to a DNS name, thus having the power to validate and revoke domains. This assumes a hierarchical structure, which enables trust to be passed from a CA to all of the certificates it issued: once a client (such as a web browser) accepts a CA as trusted, it will trust all of the certificates signed by that CA, whether they are genuine or not. This is implicit trust, which, without oversight can be (and has been) abused by compromised CAs [59, 147, 179, 190, 195].

Certificate Transparency (*CT*) [127] addresses this issue by providing a verification mechanism which requires CAs to store all the certificates they issue in an append-only log that can be queried publicly by external, independent parties, and provide proofs of inclusion. Despite requiring major changes in CA operation, it is now included in the most popular web browsers. Such a change was a gradual process, because a full roll-out of the requirement that all websites have CA certificates held in CT Logs would have broken the internet [199]. If a CA does not support CT, it is untrusted by default [185].

To verify the consistency of the certificate logs introduced by CT, gossip protocols were proposed [34, 97, 159, 203]. Due to the nature of append-only logs, revocation requires an additional workaround [183], which was formally proved [58]. Other issues include short-lived certificates in CT, and privacy-preserving proofs of misbehaviour [67]. The reliability of CT relies on auditing the append-only logs, however the number of certificates stored daily into popular logs makes it impossible for the average domain owner to act as a monitor and ensure consistency [132].

Alternatives to CT provide full transparency directly into the issuance, update, validation, and revocation of the certificates, by distributing thus reducing the centralised nature of the whole process [18, 115]. CertLedger [121] uses blockchain, with a prototype implemented on Ethereum, to show that no gossip is needed for such an approach because the blockchain is public and there is no trust being placed in a log operator like in CT. However, only CT is currently being used in practice [198]. As with any transparency tool, privacy becomes an issue [144]. There's the matter of data being released about clients to the logs as well as who is collecting and analysing these logs aside from those auditing the logs for consistency [119]. Issues with misbehaving logs was linked to mismanagement of the root store [120] or overall poor security practices [77].

2.2.2 Key Transparency

Another form of PKI maps public keys directly to human-readable names, such as a phone numbers or email addresses. This is largely used in systems where the communication takes place between end points, such as instant messaging. Automating the PKI functionality has addressed the scalability and usability issues of previous approaches and saw end-to-end encryption seamlessly deployed to the masses through secure messaging apps. These apps automated their public key management as an online service (*key server*), which in turn automated both the storage and maintenance of public key to name bindings, as well as the key lookup process. This enables users to connect to and communicate with their friends without any prior setup or need to understand public key cryptography. However, this again assumes implicit trust because trust between devices is established automatically rather than requiring it to be built manually by the users (the WoT approach), which exposes the end users to a number of attacks [213, 217].

The primary approach to providing some level of transparency for key management is manual verification. WhatsApp and Signal offer their users the option to manually verify security numbers pairwise with their friends. This can either be done by comparing 60-character alphanumeric strings, or scanning a QR-code on their peer's device if they are physically co-located [21]. However, this is an optional feature and it is likely rarely used. This process is not only tedious and error prone, but it also assumes users have an understanding of the importance of performing these checks [186]. Furthermore, verifying security numbers detects only person-in-the-middle attacks, but it would not detect ghost-user attacks since it only performs checks for the session key between the accounts and not the presence of other devices [193, 225]. The apps also allow a user to manually verify which devices have access to the account, but such a notification may be suppressed in the case of an attack, as suggested by GCHQ [131].

CONIKS [146] provides key transparency by requiring key servers to store the data in an append-only format similar to CT and allows others to publicly audit the key server's operation in a privacy-preserving way. While the CT approach has been very successful for TLS/SSL, CONIKS does not benefit from the same success because secure messaging protocols are very different by design, which means that CONIKS requires the direct collaboration of app providers to audit one another. This is difficult to achieve between closed systems such as iMessage or WhatsApp [145]. CONIKS does not deal with the multiple-devices-per-user-account scenario [146]. While CONIKS may be extended to detect whether additional keys are legitimate, this relies on the users performing a manual pairing procedure to sign new keys, which may pose usability issues for the average user.

Verifiable key directories such as those proposed by CONIKS incur significant delays when key changes occur. SEEMless [30] improves on CONIKS by reducing these delays from an hour to under a minute. However, to achieve such an improvement, SEEMless requires large amounts of storage. Furthermore, Parakeet [136] also proposes changes to the initial verifiable key directories approach, which optimises for real-world deployment by distributing small commitments to users and proposing a compaction technique to reduce the pressure added onto server storage by the growing number of keys and, respectively, users. WhatsApp took inspiration from these approaches[30, 136, 146] and recently introduced key transparency into their system by publishing an auditable key directory and offering an open source library to enable users to perform verifications [128]. WhatsApp's approach was released after this dissertation was written.

CONIKS transparency logs work in a similar way to CT logs and thus, need to be audited for consistency. One approach is to use blockchain to perform the audit [155], or Ethiks [25], which uses Ethereum to track CONIKS logs. As an alternative to CONIKS, Catena [210] uses the blockchain to track key bindings. The use of blockchain in this situation makes it an expensive process to fork this transparency blockchain. Alternatives to append-only logs focus on the use of decentralised witness cosigning for a more proactive approach of cryptographically signing (by a majority of witnesses) to attest a binding, rather than the retrospective application of transparency mechanism [118, 203].

2.2.3 Binary Transparency

Although not as popular as the previous transparency methods, binary transparency aims to provide a user with the means to verify that the binary they received from a source code repository (or app store) is the same to what everyone else received and not a special, possibly compromised version. Initial proponents saw binary transparency work in a similar way as CT: a signed binary would be added to a public append-only log, which provides a proof of inclusion that is then shipped with the binary for users who wish to install the application.

There are currently two active binary transparency logs: for the Mozilla Firefox browser [150], and for the Pixel 3 device factory images [52]. They both use CT logs to prove that the correct binary or image is sent to users. For the former, a binary hash is added to the append-only log and proof of inclusion on the Merkle-tree root is included in the binary, while for the latter image metadata is added to the log and an open-source verification tool is made available to users. Contour proposes a slightly different approach using blockchain to store hashes for binary packages [8]. It has been shown to work on over 900 000 Debian binary packages with relatively low overhead and modification to the current structure. Similarly to previous approaches, it uses the already-existing SHA256 hashes from within .deb packages to add into the blockchain and proof-of-inclusion is added inside the .deb package. Decentralising the software update process provides further transparency and eliminates the need for a single point of failure and a single point of trust [156].

2.3 Usable security

Research into usable security is widely developed with a large corpus of case studies looking at how to achieve secure usability, however secure systems still suffer from usability-related issues. In this section we review briefly the main findings of these case studies of secure systems that aim for high usability and then briefly review two use cases of PKI: PGP, which is commonly known for its usability issues, and secure messaging apps (WhatsApp and Signal), which achieve usable security.

The concept of usable security arose from the realisation that the most common security flaw of systems that are supposed to be secure is human failure [75]. This is because the systems that are designed to be secure are not always easy-to-understand nor easy-to-use by the average user, therefore users make mistakes that undermine the security offerings of the system. A common example of such usability errors is a system which requires strong computer generated passwords; it thus leads to users writing that password down on paper to have nearby, which requires additional protection to ensure it is not accessible by anyone else [184]. Conflicts are also common: the security restrictions or procedures of a system interfere with user intent or expectation, so users tend to bypass security to speed up their processes [123]. To aid usable security goals, these must be thought of in advance of creating a secure system, since retrofitting usability considerations does not work [17]. Industrial procedures ensure security risks are evaluated, but compliance with the security procedures is seldom evaluated [29]. Aside from taking into account the users of the end-product, secure systems that aim for usable security also must account for the software developers that interact with their APIs to ensure the security of the system is understood and promoted in what the developers create [87].

PGP is famous for its usability issues that led to ineffective security [182, 192, 227], which in turn limited its adoption. Two of the biggest usability-related issues are: the manual distribution and verification of public keys and the encryption/decryption process. However, even if all these usability issues were addressed, users would still face the burden of manually distributing and managing their keys.

A completely different approach was taken by modern secure messaging apps, such as Signal and WhatsApp, which thrived by automating the public key management and trust establishment process by using an online key server maintained by the app provider. These apps have successfully deployed end-to-end encrypted messaging to billions of active users, an area in which many others have failed. This enables users to connect to and communicate with their friends without needing any understanding of public key cryptography concepts. However, the need for transparency led these apps to providing the option for users to manually verify keys. This approach is faced with the same usability and scalability issues that PGP encountered [186, 219]. This is because the average user is expected to understand why they need to verify the public keys their friends have, as well as why they should regularly perform these checks. Scalability is just as important, since it is infeasible to expect users to perform and re-perform these checks with each of their friends on a regular basis. Although this very clearly causes usability issues, it does not affect the normal operation of the messaging app since it is an optional verification method.

2.4 Secure communication

This dissertation deals with secure communication. Particularly, we look at the process of establishing secure communication between end-devices and between users. We detail in this section some of the primary forms of secure communication as a foundation for the chapters that follow.

2.4.1 Pretty Good Privacy

Pretty Good Privacy (*PGP*) is one of the first programs to offer encryption to "everyone", since up until then encryption was only available to governments and large corporations [74, 235]. It offers the authentication and confidentiality of both files and emails. It uses RSA public-key cryptography to enable encryption/decryption and signing/verifying to take place. Beside RSA, PGP also now supports Elliptic Curve Cryptography (*ECC*) [5, 108]. Instead of a PKI it uses WoT, in which users hold the responsibility to manage their own keys, which includes creation, revocation, and distribution of keys to friends. Section 2.3 discussed the usability and scalability issues that PGP users encountered, which lowered adoption [74, 182, 192, 227].

2.4.2 Off-the-record messaging

Off-the-record communication (*OTR*) [26] was presented as a solution to enable end-to-end security to instant messaging protocols since protocols such as PGP [74, 234] and S/MIME [63], which depend on long-term keys, are less suited for such communication because they do not provide deniability. Aside from offering confidentiality and authentication, properties in common with its predecessors PGP and S/MIME, OTR also offers perfect forward secrecy and deniability, such that if key material is compromised, the system can recover to a safe state and the read messages cannot be attributed to a certain user.

OTR was initially shown to be insecure due to the insecure key-exchange protocol and design choices [54], which was fixed in later iterations. Another issue with OTR is that it only enabled secure conversations to take place between two users, however enabling the same between multiple users is non-trivial. One approach to enable such multi-party conversation to take place is to designate one of the users as a host (or "virtual server") [20], which places a large amount of trust on the host. An alternative change to OTR to enable multi-party conversations is to provide pair-wise authentication between the group participants [82]. A more involved group key agreement protocol uses the concept of a "circle of keys" wherein a shared secret can be computed by anyone with a private key corresponding to a public key in the circle of keys [133]. OTR requires both participants to be online; a solution which addresses these limitations proposes making available a chain of session keys to users, which would enable re-keying (for the duration of the chain) without needing both users to be online [70].

OTR also suffered from similar usability issues like PGP because it expects users to understand and securely use cryptographic concepts as seen in the Pidgin instant messaging application [201]. Improvements based on the Socialist Millionaire's Problem aim to resolve such issues [9], which replaces the need to understand keys and fingerprint verification with a password or a passphrase, concepts with which users are familiar.

2.4.3 Secure Messaging

Secure messaging bridges this usability gap by automating the trust establishment process involved in exchanging the long-term public keys as well as performing any session key calculation and renewal. The former is enabled through the use of an online key server as PKI. This allowed apps such as Signal, iMessage and WhatsApp to deploy end-to-end encryption to billions of active users without requiring any specific knowledge. Other apps, such as Facebook Messenger and Telegram also offer end-to-end encryption but it is not enabled by default.

Despite all the usability improvements and the wide-spread use of end-to-end encryption in daily life now, secure messaging apps still encounter usability limitations. The most evident one is linked to the optional key verification mechanisms offered by Signal and WhatsApp, which consist of expecting users to perform pair-wise checks with all their friends and compare a long alpha-numeric string (or scan a QR-code on their friend's device if nearby). This process has several limitations [95, 186, 219]: (1) it expects users to understand the importance of performing such checks as well as re-perform them regularly; (2) it is error prone since it is a manual comparison, and (3) it is optional so it will only be performed by the more privacy-focused users. An alternative is to create a link between online identities, such as secure messaging accounts linked to social media accounts, which would then enable automatic authentication, thus bypassing this manual process [89, 218]. However, such an approach raises privacy concerns due to the amount of data collection typically performed on social media platforms.

Other limitations include the loss of privacy protections (or downgraded protections), particularly due to the centralised nature of these apps. Firstly, most of these apps require users to share their phone number or email address in order to sign up. This, in turn, helps the automated trust establishment process by giving other users a way to identify their friends using a unique human-readable name. Secondly, despite offering end-to-end encryption, there is no metadata privacy, which allows the service operator to collect information about who is messaging whom and when. Such information, linked to the database of user identifying information gives significant power to the service provider. New approaches are emerging to decentralise this well-established architecture and providing enhanced security features such as anonymity or deniability [214]. For instance, Session [107], offers unique identifiers without having to link them to personally identifiable information of the user, as well as metadata privacy by providing onion-encryption of the messages and routing them through the Oxen Service Node Network [165]. Biometrics-driven approaches are also explored, which ensures that private keys do not need to be stored on device anymore since these are derived from biometric characteristics of the user, and an associated public key is generated using an asymmetric fuzzy encapsulation mechanism [222].

A security evaluation framework proposes assessing secure messaging tools on three levels: security, usability, and ease-of-adoption [213]. This framework identified several open problems in the secure messaging components: trust establishment, conversation security, transport security. The Electronic Frontier Foundation (*EFF*) created a secure messaging scorecard [151] to evaluate the secure usability of messaging apps based on seven criteria. Aside from the usability issues, one of the significant observations the EFF made is that most users do not need a "bunker" for their conversations, they need a system with which security-focused users are happy to use, but which can also be used by regular people.

2.4.4 Secure collaboration

There are many other forms of sensitive data exchange, beyond emails and instant messaging, that require additional consideration to end-to-end encrypted protocols. Secure collaboration is one such example in which different users with one or more end-devices want to securely collaborate on a sensitive document. More specific examples include journalists sharing a large database of files to report on a common story [142, 143], or lawyers collaborating on privileged documents with other lawyers and their clients [130]. Applications that currently enable such activities are not end-to-end encrypted due to the dependence on protocols such as Operational Transformation (*OT*). They use encrypted data, but OT relies on a central server to merge concurrent modifications and to relay current status to the users (e.g. if one device is offline for an extended period). This means that communication is only encrypted while in transit to and from the server and is decrypted on the server for the merging process. Such a centralised approach assumes a significant level of trust in the server operator. By adapting the protocols used by secure messaging, we enable the creation of end-to-end encrypted collaborative applications [116]. This approach helps us retain important security guarantees, such as confidentiality and integrity in the face of network attackers and malicious servers. However, insider threat from malicious collaborators is non-trivial to address.

2.5 Anonymity

Modern cryptographic protocols offer communication privacy over unsecured channels, such as the Internet. However, the metadata from these messages is exposed by default. Through analysing metadata, an observer can learn per-communication details such as who is messaging whom, how long messages are (or duration of a call) and what time the messaging occurred; or learn user patterns by looking at the traffic volumes over time, which can lead to re-identification [98]. The level of anonymity differs depending on need, ranging from hiding

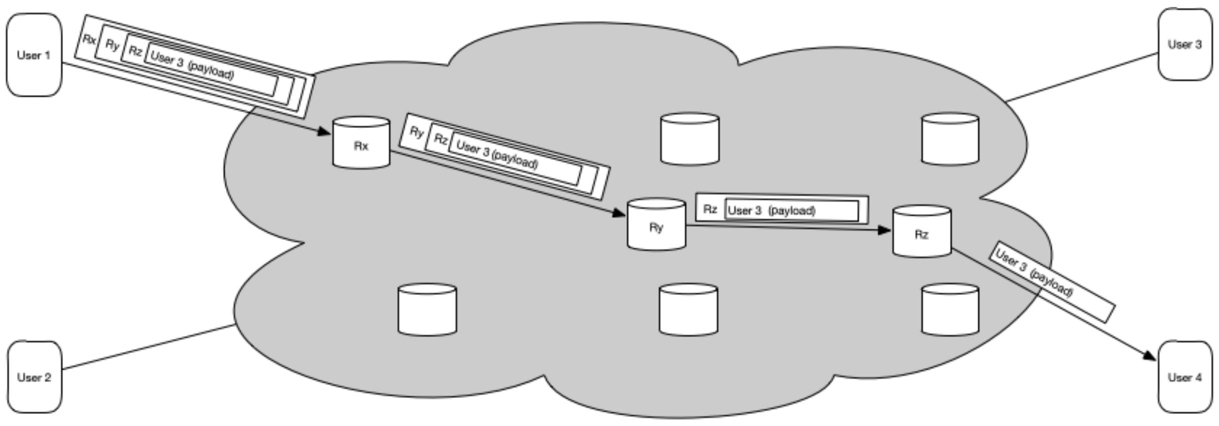


Figure 2.1: Onion routing diagram when User 1 sends a messages to User 4

from an observer with whom the user is communicating, to completely hiding whether any communication is taking place. We discuss in this section some of the most common approaches to achieve anonymity and what these anonymise.

2.5.1 Onion routing and Tor

Onion routing [83, 175] is the process of providing source routing using nested encryption, which means that the sender determines the full path through the network until it reaches the recipient. Figure 2.1 shows how this is done by applying multiple layers of encryption to the packet, which is then forwarded through a set of relay nodes in the network, such that the outermost layer is always the next remaining relay point and the innermost layer is the encrypted packet for the recipient. This approach offers protection for both eavesdropping and traffic analysis. If any of the intermediary nodes is corrupt, the packet is still safe since they cannot trace the communication back to the sender. This property also holds if the recipient is corrupt, since they are unable to trace the communication back to the sender.

Onion routing is circuit based in order to maintain efficiency. This means that the sender establishes a path through the network (circuit) by performing handshakes with intermediary relay nodes, which allows it to create session keys, used by all the packets within a session, which use the same path. This creates linkability of packets for all those packets in the session and enables fingerprinting of traffic. Because there is no added latency in the onion routing process it makes it vulnerable to timing attacks.

Tor [57] implements onion routing, but differs from the initial proposals by implementing several improvements such as perfect forward secrecy and location-hidden services. Despite proposing a reasonable trade-off between anonymity, usability and efficiency, Tor has seen criticism over usability issues. In particular, despite being a low-latency network, Tor users commonly experience high delays, particularly visible since most Tor usage is for Web applications [53]. Part of the reason for these delays has to do with delays on the Tor nodes, and one mitigation is to detect in advance the nodes that lie about their capacity [167]. Another usability issue identified within the Tor system is the reliance on the need for users to understand the system, since without proper usage [73, 158] or low adoption numbers [104] it can lead to deanonymisation.

2.5.2 Mix networks

Mixing traffic is not a new concept and it was initially proposed under Chaum anonymous communication network in the early 80s [32]. It is the predecessor to onion routing and although similar in concept, there are some essential differences. For instance, mix networks are packet based, such that every packet has its own individual route, thus offering unlinkability at the packet level. This approach however, is more expensive than onion routing due to the extra public key cryptography operations required per packet.

Despite the encrypted package looking different between entering a mix node and exiting a mix node, since the layer of decryption is stripped to find the next stop in the path, therefore changing the bytes, such an action is still subject to timing attacks [191], which would enable an attacker to guess the path of a packet based on the correlation between entering a mix server and exiting it. To mitigate timing attacks, each server holds packets for a set threshold and then releases them when the threshold has been met.

One common approach to improve the anonymity is to add extra (*dummy*) traffic into the network. Such an approach adds noise into the network, increasing the anonymity offered by aggregating into larger anonymity sets. Furthermore, adding dummy traffic into a mix network contributes to reducing the overall packet latency in the network.

There are several types of mix networks with different anonymity guarantees based on the chosen approach. For instance, mixnet-based proposals Loopix [169] and Vuvuzela [215] use dummy traffic to incease the anonymity set, however leave the dummy traffic strategy as future work. Nym [160] is a Loopix-inspired system, which implements a multi-purpose source-routed mixned design with Nym Crypto Tokens used as incetives for used to make the mixnet decentralised. Mixmaster [44] and Mixminion [48] are examples of mixnets that sacrificed latency to maximise anonymity, thus making them unusable for situations that involve several messages being sent back and forth, such as web browsing or instant messaging.

2.5.3 Covert channels

Providing a covert channel on top of already existing communication technologies is one approach to reducing communication metadata and providing anonymity, without requiring a large number of active (or knowingly-involved) participants. For instance, a user's browsing activity can be used as the required layer of communication, such as hiding messages in HTTP request headers [19, 194]. Sending a constant amount of data per epoch like in constant throughput channels is another common way to provide a covert channel that provides anonymity [41]. Such an approach does not require a large number of active participants, however, it is usually limited by bandwidth and latency since imposing large data traffic on users who are not actively using the protocol causes concern, and also drives the number of real transmissions per epoch per user. Typically, in this approach users and servers interact through dropping encrypted messages for each other in generic mailboxes, from which all the users can fetch regularly [11] or broadcasting encrypted messages to all the users [39].

2.5.4 DC-nets

The aim of DC-nets is to provide sender and recipient untraceability in a network by using a broadcast-based approach [31]. In this solution, once a group is established there is no need for the participants to interact. However, in this approach the users are effectively running the infrastructure, so for the untraceability properties to hold, all the group participants are expected to also broadcast a message every time a message is sent. Such an approach encounters challenges with reliability as seen through several implementations of DC-nets [40, 81, 232], most of which are vulnerable to disruptive participants that block communication, in part or in full. In an attempt to correct this, Verdict [42] provides an implementation of verifiable DC-nets that uses retrospective blaming to proactively enable the tracing of disruptors in a group.

Although modern DC-nets make use of client servers to make the process more efficient, there are still very high communication overheads. Lastly, detecting cheating participants in this type of protocol is non trivial, with solutions involving multiple broadcast rounds that include a set of "traps" for the cheating participants [31] or single broadcast rounds [84].

2.5.5 SecureDrop

SecureDrop [188] is a system created and maintained by the Freedom of the Press Foundation to enable whistleblowers to submit stories and materials to media organisations in a secure way. Media organisations have to set up and run two SecureDrop servers in order to accept documents from sources anonymously. The first server is the document receiving server, which is made available as a Tor Hidden Service, and the second server performs security monitoring of the first. The media organisation then have an air-gapped environment which is used for decrypting the encrypted messages that have been received through the Tor Hidden Service and stored locally on an encrypted drive.

In order to use SecureDrop, sources have to install a Tor browser, navigate to the media organisation's Tor page and start a SecureDrop session for which they are given a code name. They can use the code name every time they need to check their SecureDrop inbox for any new messages, or send a message. For the media organisation, each user (account) is identified by an unrelated code name. Although enabling high security protections for sources, it is difficult for sources who are not technically-savvy to use SecureDrop [55]. Journalists are provided with a SecureDrop Workstation, which allows them to see submissions, respond to sources and track associated documents [189].

From an anonymity perspective, SecureDrop addresses the problem of metadata associated with most end-to-end encrypted messaging systems, which is a give-away for such high profile scenarios as whistleblowing. SecureDrop has independent security audits before every major release and up to 2018 it had five independent security audits, which identified issues that have been addressed in subsequent versions.

2.6 Summary

This chapter introduced the background and current research to support the rest of this dissertation. We provided a brief historical review of encrypted communication, covering main events, prior to introducing the problem of trust establishment and discussing two main approaches: Public Key Infrastructure and Web of Trust. In discussing trust establishment, it is necessary to review the level of transparency needed for users to trust that services operate correctly, which we discuss in Section 2.2 and is an essential background to the work in Chapters 3-4. We also state that systems are only as secure as they are usable and discuss this with famous examples of both failures and wins in Section 2.3. We then provided a longer summary of secure communication, which is central to this dissertation. Last, we discuss several approaches to provide anonymity of conversation, essential to our work in Chapters 5-6.

CHAPTER 3

DETECTING KEY SERVER EQUIVOCATION IN END-TO-END ENCRYPTED MESSAGING

The Crypto Wars [105, 125] have seen numerous requests for exceptional access to devices and encrypted systems. The usual motivator? – support for law enforcement and intelligence agencies in their jobs to thwart terrorist attacks and maintain national security. Most proposals include some form of manipulation to defeat cryptography: the introduction of backdoors (e.g. ensuring pseudorandom number generators in elliptic curve cryptography are predictable [92], or lobbying for small-enough key sizes in encryption standards to ensure they are easily crackable [38]); or the use of frontdoors (e.g. the failed attempt at introducing government approved chipsets [168], or decentralising key escrow such that the government holds half of the encryption key and all they need is a warrant to obtain the other half from the company [100]). However, these requests were disputed because they would introduce significant risks to the communication security and privacy offered to end-users by opening extra attack surfaces [174], or simply facilitating unlawful mass gathering of data without any oversight [90, 93, 124], thus breaching human rights.

This chapter describes *ghost-user attacks* – a means for law enforcement or national security departments to add additional covert devices to a communication channel.

We research methods to obtain cryptographic proof that such an attack took place [217]. Ultimately, we propose a gossip-based approach to monitor the consistency between what updates the key server provides about a user's account and what the devices belonging to the user acknowledge. This approach enables us to differentiate between the key changes that happen on a user's account and, in turn, detect ghost-user attacks and dismiss any false positives. We analyse our approach's security and privacy, and effectiveness in Chapter 4.

The content of this chapter consists of ideas drawn from a position paper in which we discuss key change events and why the attack is possible, coin the term "ghost-user attack" and discuss suitable cryptographic proof for detecting such attacks, which we published in 27th International

Workshop on Security Protocols (SPW) [217]. In this chapter we build on these initial ideas and propose a gossip protocol to detect ghost-user attacks. The initial idea for the attack originated from discussions with Alastair on the subject. I was the primary author in this publication. My collaborators Daniel and Martin, together with our supervisor, Alastair, have helped with lengthy discussions about the attack, key changes, protocol structure, and analysis. They have also provided invaluable feedback on the write-up. Throughout the research I led the protocol design, the formulation of KNGs to allow us to make the protocol suitable for a variety of apps, as well as the analysis that follows in Chapter 4.

3.1 End-to-end encrypted messaging

End-to-end encryption is now default for billions of users in popular secure messaging apps, such as iMessage, WhatsApp, and Signal. In such apps, data is encrypted on a sending device and only decrypted on the receiving device, flowing encrypted through any servers in between.

End-to-end encryption is typically implemented with public-key cryptography: each device generates a long-term public-private key pair and ensures that the private key is kept secure. Then, the public-key and a unique human-readable name (hereafter *name*), such as a phone number or email address, are shared with the app provider. The app provider maintains Public Key Infrastructure (PKI) as an online service, called a *key server*, that maps names to public keys. For example, when Alice wants to send a message to Bob, her device first asks the key server for the public keys of Bob's devices and then encrypts her message using the keys provided by the key server. Provided that Alice receives Bob's keys (*and only his keys*) from the PKI, end-to-end encryption prevents the rogue employee, cyber-criminal, or government agent from decrypting her message.

This is the working model of secure messaging apps. In this design, trust moves from the communication server, which now only sees encrypted data, to the key server, whose only purpose is to enable trust establishment between end devices by facilitating key lookup. A compromise at the key server allows the attacker to add or modify the public keys associated with names. For example, because end-to-end encrypted messaging apps accept multiple devices as end-points on users' accounts, an attacker may associate an additional mobile device with Bob's account. Then, when Alice encrypts subsequent messages and sends them to Bob, her messages will be received and readable by the attacker. This is because all messages sent by these users travel via the messaging server and arrive at all the devices registered on the two accounts involved in communication. Furthermore, neither Alice nor Bob may be aware that this has happened. This is the strategy proposed by GCHQ to wiretap end-to-end encrypted messages without modifying any of the encryption [131] and is called a *ghost-user attack* [217].

Modern end-to-end encrypted messaging is successfully used by billions of people largely due to automation of trust establishment and message encryption. This automation however, increases the importance of operator accountability in end-to-end encrypted messaging, to prevent widespread state surveillance of personal communication [213]. In practice, WhatsApp and Signal provide this through pair-wise manual verification: users can either compare a short alphanumeric string, or they can take a picture with their smartphone of a QR-code displayed on another user's smartphone. This is optional since both apps run a central PKI. We are not aware of any research quantifying the frequency of these manual key checks. It seems likely that this feature is rarely used because it is error prone [186], provides no obvious, immediate benefit to the user, and requires users to regularly undertake a manual process with their friends. Alternatives include decentralising power by displacing and reducing the trust placed in certificate authorities [203], and encouraging certificate authorities to submit the contents of their PKI to public append-only logs in order to support external audits [127, 146]. These solutions aim to detect misbehaviour after-the-fact. The main challenges of existing approaches include: interoperability between end-to-end encrypted services (closed vs. federated), the requirement for operators to cooperate and share the contents of their key directories, and the requirement that users perform manual key checking. WhatsApp also recently announced the release of key transparency through verifiable data structures [128], an approach inspired from prior work such as CONIKS [146], SEEMless [30] and Parakeet [136].

We provide a novel, automated verification method that can be performed in parallel with the normal operation of the key servers. Our approach combines the benefits of automation, which takes the cost of verification away from the end-user, and Web of Trust style distributed key checking, which takes advantage of a diversity of network paths typically seen by mobile devices. Our design for the gossip protocol allows us to perform automatic verification of the bindings between public keys and names provided by the key server. This is the first protocol to focus on detecting ghost-user attacks through key verification.

We choose gossip protocols because of their scalability and resilience due to the inherent redundancy they produce in the network. Furthermore, they are easy to implement [129] and platform and network topology-independent, which makes them highly versatile. Using a background data dissemination gossip protocol [22] allows us to continuously adjust the confidence in the correctness of a public key-to-name binding. Here, propagation latency is not critical since such latency only affects speed of verification, not speed of message delivery.

Apps bind public keys to names differently. We therefore, generalise the bindings into a Key-Name Graph, hereafter *KNG* (Section 3.2). Tracking the changes in the KNGs enables us to differentiate between benign and malicious key change events, reducing false positive notifications (i.e., notifying users of legitimate key-change events, which can take attention away from actual attack cases). Security is then provided by checking that the key server is distributing the same KNG for a user's friend as the KNG discovered using our gossip protocol. By sharing the KNG and the cryptographic proofs of the edges in the graph over local networks, users can constantly check their key bindings with their friends, providing an alternative mechanism for verifying the correct operation of the key servers. This proposal is practical and compatible with the normal operation of messaging apps.

The base protocol provides gossip opportunities for users who frequently co-locate with their friends, or those who have friends in common, thus restricting gossip reach. We detail the type of changes needed from the key server to prove misbehaviour to third parties (Section 3.4.6). We also discuss two alternatives that enable wider gossiping with strangers (Section 3.4.7): friends-of-friends and epidemic-style gossip.

Our approach provides significant security above that provided by a central server since an attack against our proposed solution requires the attacker to compromise the victim's end device or *all* the local networks the device ever visits. Furthermore, its simplicity coupled with support for different underlying network technologies makes it suitable for deployment in many existing apps and does not necessitate changes at the key server, such as generating or circulating signatures.

We make the following contributions in this chapter:

- an analysis and subsequent formalisation of public key-to-name bindings into KNGs that can represent all current combinations of bindings in production end-to-end encryption apps (Section 3.2).
- a novel baseline gossip protocol for key verification, which combines the benefits of automation and web-of-trust style distributed key checking (Section 3.4.4).
- an analysis of the type of key-to-name binding changes that affect a key-name graphs (Section 3.4.5).
- a proposal for changes at the key server which would allow a user to prove misbehaviour to a third party (Section 3.4.6).
- a proposal for changes in the baseline gossip protocol to enable gossiping with strangers (Section 3.4.7).
- a discussion of possible underlying network technologies suitable for the gossip protocol's deployment (Section 3.5).

3.2 Key-Name Graphs

Associating public-keys with people is surprisingly difficult. Commonly, public-keys are bound to a human-readable name to make it easier for humans to identify other users. We formalise these bindings to a *KNG* (Figure 3.1). While the end-to-end encrypted apps available to users seem different in their setup on the surface, we provide a walk-through of some of the different designs and produce a model to bind names to public keys that will work for all current apps.

This general model will be used in the gossip protocol (Section 3.4.4) and it can be applied to both current and new end-to-end encrypted platforms.

Signal and WhatsApp used to map a single name to one device (one public key). Now, they support multiple devices for the same account: browser log-ins (WhatsApp) or desktop app (both). The Sesame Algorithm [137] provides this multi-device setting. With Sesame, devices either share the same key-pair ("per-user identity keys" – WhatsApp in the old model), or each device generates its own key-pair ("per-device identity keys" – Signal and WhatsApp now). In WhatsApp, without an active Internet connection for the mobile phone the other devices won't receive messages, while in Signal, each device can independently receive messages and decrypt them with its own key. The keys map to a single "User ID", which is the user's phone number.

In contrast, iMessage [12] supports not only multiple devices per user, but also multiple names (e.g., email address, and phone number) and these are associated with a set of three key pairs for each device: an RSA 1280-bit key, an encryption EC 256-bit key on the NIST P-256 curve, and an ECDSA 256-bit signing key. The private keys from the key pairs are associated internally with the devices on which they have been generated using the device's Apple Push Notification (APN) address.

We formalise such key bindings for an individual as a *KNG* to refer to the undirected tripartite graph formed by a many-to-one connection from the names set to an account identifier, and a one-to-many connection from the account identifier to the set of public-keys as seen in Figure 3.1. Each device maps to exactly one set of key pairs through a one-to-one mapping. Each of these sets may contain multiple key pairs per device (iMessage), a single key pair shared across devices (WhatsApp), or anything in between. Our KNG framework supports all the configurations found in end-to-end encrypted apps today and can be used to model the names and keys used in the iMessage platform as well as Signal and WhatsApp.

3.3 Adversarial assumptions

In a successful ghost-user attack, the attacker receives copies of *all* the messages Alice sends or receives, without either participant knowing. We describe the actors involved, analyse an attacker's goals and capabilities and review our assumptions before formalising our adversary model.

3.3.1 Actors

A messaging app has the following actors: users, key server, messaging server, and app store.

Users are the main actors of a messaging app. Alice is a user of the app. She begins by installing the app on her device, typically from an app store. She then registers to use the app by providing her *name* (phone number or email address). She uses the app to securely message her friends.

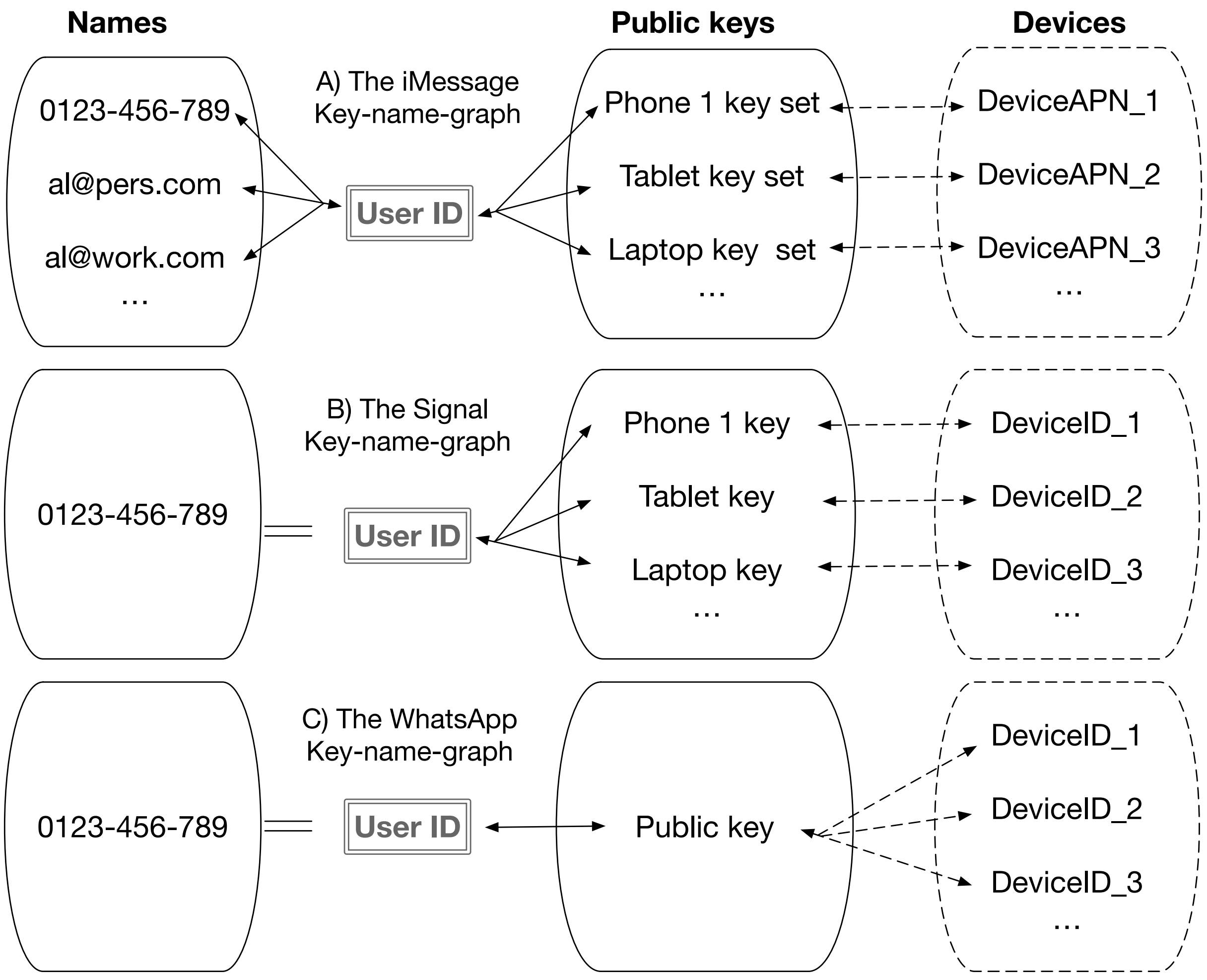


Figure 3.1: An example KNG for Alice for A) iMessage, B) Signal and WhatsApp (current), and C) WhatsApp (old model). These show the versatility of formalising bindings into KNGs. A) there are many to one links from Alice's different names for her account to an internal identifier, and one to many links from the internal identifier to all of the public key sets that reside on Alice's devices; each of these key sets map to exactly one device APN address [12]. B) and C) use Sesame [137] and have one name (phone number), which removes the need for an internal identifier. The name maps to either a public key for each device (B), or to one key, which then maps to all of the devices (C).

We define a *friend* as another user of the messaging app, whose name is recorded in Alice's contact list on her device. When Alice has *more than one device*, all of these devices have the app installed and have access to all the message contents (gained through authentication means defined by the app itself – more details in Section 3.2). When she uses one of these devices to communicate with a friend, all of her devices receive the message Alice authored (from either of the devices), or that her friends have sent to her. Upon installation, devices generate one or more public-private key pairs depending on the app implementation (Section 3.2).

The *key server* is an online service maintained by the app provider. It serves to automate the storage of public key to name bindings and facilitates key lookup. The key server, depending on the app, can store more than one binding for a name, which means that users can have more than one device connected to their account (for details see Section 3.2).

The *messaging server* is also an online service maintained by the app provider. It ensures message transmission between different users by using names to identify the destination.

The *app store* is an online service, independent from the app provider, offered by different mobile platforms (e.g. Apple App Store and Google Play Store). The app store serves upon request the messaging app binary to users' devices for installation or updates.

3.3.2 Assumptions

The most common devices with access to end-to-end encrypted messaging apps are smartphones, tablets, and laptops. We focus on these end-user devices which have access to the end-to-end encrypted messaging app and describe our main assumptions below:

Secure corresponding devices. We assume that the corresponding devices are free of spyware apps, including root spyware, and that no adversarial possession or physical attacks take place. This excludes from our threat model the chance for message contents to be extracted through other means, such as unlocked device screen capture, keystroke logging or UI control.

Secure encryption. We assume that the public-key algorithms used for encryption in the messaging app are in a secure state. We consider backdoors or flaws in encryption algorithms out of scope for the correct operation of the end-to-end encrypted messaging app.

Secure app store. Attacks on the app store are outside of the scope of our threat model. Even if an attacker were to compromise the app store that delivers the app's updates, they won't be able to deliver a compromised version of the app without gaining access to the keys used for signing the app, which are typically stored offline.

Physical mobility of devices. We assume that users sometimes meet *some* of their friends on a local network and that the rate of meeting people is higher than the rate of device change. We justify these assumptions in Chapter 4.

Time synchronisation. Smartphones use the mobile network to synchronise their clocks; tablets and laptops usually have an active Internet connection, so they can use services such as NTP to maintain time synchronisation. Since key change events are expected to be rare, we assume a loose time synchronisation of approximately five minutes is sufficient.

3.3.3 Attack goals

The attacker's objective is to read the messages sent between Alice and her friends. We discuss two ways in which an attacker can achieve their goal: *person-in-the-middle attacks* and *ghost-user attacks*. In a person-in-the-middle attack the attacker replaces all of Alice's keys with their own at the key server, compromising the messaging server to receive all the messages intended for Alice before re-encrypting and forwarding to Alice's devices. In a ghost-user attack, the attacker adds an extra device (i.e., a new public key-to-name binding) to Alice's set of devices recorded by the key server, thus attaching themselves to Alice's account. The attacker now receives a copy of all the messages Alice sends or receives without the sender or recipient knowing.

A person-in-the-middle attack is fragile when compared to the ghost-user attack. The person-in-the-middle attacker has to compromise all of the keys Alice has in her KNG and update all the keys previously shared with her friends. In this case, the full set of key-to-name bindings of the user changes, triggering a key change notification in some end-to-end encrypted apps. For a ghost-user attack, an additional key is inserted into the set, which does not notify any of Alice's friends. In some apps, for instance iMessage, Alice herself would be notified on her other devices that a key has been added. Our solution improves on this by enabling detection of the event solely by the devices, supporting users who only have one device, and also notifying more than just Alice herself, but also her friends. Furthermore, the attacker would also have to compromise Alice's friends devices to avoid detection.

In addition to detecting ghost-user attacks we also detect person-in-the-middle attacks since these also appear as a key change. However, in order to detect them, friends have to observe the correct keys before the person-in-the-middle attack starts. An advanced user might detect an initial person-in-the-middle attack by noticing that gossiping is not working when it should. This case occurs when the user is unable to confirm the keys of his friends on the local network, since the user does not have any correct keys for his friends.

The underlying problem, which stops users from detecting person-in-the-middle attacks, is the large number of key-change notifications users are currently exposed to. Because of this, users are likely to ignore a key-change notification since often this is harmless, such as their friend reinstalling the app or purchasing a new device. Our work aims to reduce the signal-to-noise ratio of these key-change events so that an alert is only issued when either a person-in-the-middle or a ghost-user attack is discovered.

3.3.4 The adversary model

Thus, we assume the following adversary model:

1. The adversary can compromise the key server. We assume the attacker can access, modify, inject, and delete bindings for some or all the keys in the key server. In doing so, the messaging server will send a copy of all future communications between the attacked user and their friends to the attacker's device.
2. The adversary can compromise the messaging server. However, without access to the correct private key the adversary is unable to break encryption and access the messages.
3. The adversary cannot compromise the app updates supplied by the app store.
4. The adversary can track devices on a local network, but has limited visibility over local networks, such that it can only access, modify, inject, or delete messages sent on a subset of the local networks to which a user connects.

3.4 The gossip protocol

We propose a gossip protocol to enable Alice's friends to continuously verify that Alice's KNG provided by the key server, is the same across all her friends and matches the KNG locally gossiped and signed by Alice's devices. This protocol eventually detects ghost-user attacks and person-in-the-middle attacks (Section 4.1.4) by differentiating between the types of key changes that can take place in the app. The gossip protocol can be used with a variety of underlying local network technologies (Section 3.5), however, technology-specific details are omitted from the description of the protocol for simplicity.

3.4.1 Functionality goals

We define below the main functionality goals of the gossip protocol:

F1: automated detection. The gossip protocol enables automatic detection of ghost-user attacks and person-in-the-middle attacks.

F2: minimal changes. The inclusion of the gossip protocol in an existing end-to-end encrypted messaging app requires minimal changes to the support infrastructure (i.e. key server or messaging server).

F3: flexible underlying network. The gossip protocol does not restrict an underlying network specification as it aims to be flexible regarding the choice of underlying local network technology. Therefore, it is designed to work with a wide range of networking protocols such as WiFi, Bluetooth and 5G. Further information regarding compatibility of these technologies is discussed in Section 3.5.

3.4.2 Security goals

We outline the protocol's security goals with reference to our adversary model (Section 3.3.4) and justify in Section 4.1.4 how the gossip protocol achieves these goals.

S1: correctness. The adversary cannot mask their access to an account as a normal key change. The protocol correctly identifies all the types of key changes for KNGs (Section 3.2).

S2: soundness. The adversary's presence cannot be hidden indefinitely. The protocol eventually detects ghost-user attacks when one has taken place, provided that the user successfully gossips with at least one friend post-compromise.

S3: availability. The adversary cannot completely stop a user from gossiping on local networks. If the ability to gossip is enabled in the app, the user's devices *can* gossip.

S4: privacy. The protocol maintains the privacy protections already offered by the messaging app.

3.4.3 Limitations

The gossip protocol's main limitation is its dependence on human mobility. While cases where users do not meet with friends on local networks for extended periods of time do not affect the normal operation of the end-to-end encrypted messaging app, it does make for windows of vulnerability of varying sizes. The gossip protocol aims to improve on the current established manual key verification process and provides extension opportunities past the immediate friends network (Section 3.4.7).

The gossip protocol robustly detects ghost-user and person-in-the-middle attacks while minimising false positives from legitimate key change events (e.g. new device). However, investigating how to usefully report this information to the user is out of scope for this paper.

The protocol detects these attacks, however, it does not enable these attacks to be proved to a third party, as that would require changes at the key-server to sign records it supplies (Section 3.4.6), which we left out-of-scope by focusing only on app-level changes.

Setup: We describe the knowledge base for Bob's device.
***Note:** Throughout Alg 1-4 we denote procedures with psudocode in* SMALLCAPS *and those without in* `teleType`.

1: *precmpHashToDeviceMap* – map of public key hash to deviceID
2: *publicKeys* – map of friends' deviceID to public key
3: *localAdsList* – list of adverts currently on the local network
4: *pendingRemovals* and *pendingAdditions* – temporary list of removals and additions not yet confirmed by the key server stored for each friend

Algorithm 1 (*Advertise*): Alice's device creates an advert for the local network using the public-private keypair on the device.

```
1:  procedure ADVERTISE(KNG)
2:      t ← getCurrentTimestamp()
3:      broadcast(GENSIGNATURE(KNG, t))
4:
5:  procedure GENSIGNATURE(KNG, t)
6:      h ← hash(KNG)
7:      ttl ← 24 hours
8:      sig ← sign(privKey, h||t)
9:      hint ← hash(hash(pubKey)||t)
10:     return (sig, hint, t, ttl)
```

3.4.4 Baseline gossip protocol

Manual verification, currently available in some end-to-end encrypted apps, assumes a three-step process: (1) two users decide on a location to meet in person (*advertise*), (2) find each other at that location (*discover*) and, (3) confirm the keys on their devices are correct by scanning each others' QR-codes (*sync*). Our approach is inspired by this process but the protocol enables us to automate these three steps while leveraging decentralisation and human mobility. This removes all manual labour for the common case where the KNGs match, and identifies attacks in the case where the KNGs do not match through cryptographic evidence. We aim to improve on the established manual process and enhance transparency for messaging applications, while limiting the changes to app-level with a protocol that is infrastructure agnostic.

The baseline gossip protocol operates by constructing gossip messages that are readable only by those who know the user's public key and have previously constructed their own copy of the user's KNG (i.e., the user is a friend of theirs). Its aim is to continuously verify the information received from the server against the information received from gossiping about a user and maintain freshness in the common case where these two sources of information provide the same KNGs. We deal with key change in Section 3.4.5. We describe below the three main steps in the baseline gossip protocol between two friends – Alice (the advertiser) and Bob (the verifier):

Advertise. Devices advertise their presence on local networks so friends' devices can find them. For example, Alice's device advertises on arrival to a local network. It also re-advertises at regular intervals if the device has been on the same local network for a long period of time. Such intervals that can be adjusted to cater to timing constraints imposed by the underlying network technology. This process provides regular signed adverts from every device on the local network.

To build an advert (Alg. 1), the device first takes a hash of Alice's KNG, which is represented as an alphanumerically-sorted string of triples (name, public key, device id). The current timestamp on the device is appended to this hash and signed with the private key stored on the device (Alg. 1 – line 8).

A *hint* is then prepared by hashing the hash of the public key appended with the same timestamp (Alg. 1 – line 9). This hint reduces compute time in the discover phase (see below) by replacing a potentially large number of signature verifications with the same large number of hashes which are computationally much cheaper to perform [223]. At the end of the hashing process, a device discovering adverts on the network can detect whether any of the adverts were signed by a public key known to them. Because the public key hash is hashed with the current timestamp, the hint does not act as a constant identifier. A similar approach of sharing encrypted packets between devices in vicinity of other devices has been used by Apple/Google in the SARS-Cov2 pandemic for automatic contact tracing [101].

Adverts broadcast on the local network consist of an opaque signature and hint since those without the correct information cannot verify them, and the cleartext timestamp and a Time-To-Live (TTL). The use of the one-way hash prevents both data modifications and supports security goal G4 by not exposing any static identifiers or other forms of identifiable data on the network, because the hints and signatures are opaque without knowledge of the public key. A TTL ensures the adverts do not persist indefinitely when the underlying network can cache data (in Algorithm 1 this is specified as a 24-hour constant, however it can be tailored depending on the network's ability to cache this data).

Discover. Devices regularly check adverts they see on a local network to identify those owned by friends and initiate the sync process. Every device holds a pre-computed list of public key hashes belonging to the devices owned by the user's friends. On every run, the device gets a list of recent adverts placed on the local network and attempts to link them to the list of friends' devices' keys by using the timestamp provided by each advert, as shown in Alg. 2 – lines 3:7.

The discover protocol returns a list of adverts that belong to the Bob's friends, which will be passed to the sync phase.

Sync. Every advert received from the discover phase belongs to a device owned by a friend. The sync phase enables devices to update their knowledge of a friend's KNG. Updates from gossip happen either if the KNG has remained unchanged from the previous gossip (most common case), or if there are changes (rare cases, handled in Section 3.4.5). Refreshing knowledge of

Algorithm 2 (*Discover*): Bob's device identifies which adverts belong to his friends from a list of local adverts. Returns *discoveredAds* – list of all the discovered pairs (advert, signing key hash) that match a key known to the scanning device.

```
procedure DISCOVER(localAdsList)
    $discoveredAds \leftarrow \{\}$
    for each $ad \in localAdsList$ do
        for each $h \in precmpHashToDeviceMap$ do
            if HINTMATCH(ad.hint, h, ad.t) then
                $discoveredAds$.put($ad, h$)
    return $discoveredAds$

procedure HINTMATCH(adHint, hashPub, t)
    $H_t \leftarrow$ hash($hashPub||t$)
    return $H_t == adHint$
```

KNGs even if they remain unchanged is important to verify the consistency of correct key server updates over time [217].

Sync between Alice and Bob is performed in one of two ways: DIRECTSYNC, when Alice's device is available to connect on the local network and confirm any changes *directly*; and SECONDDEGREESYNC (provided the underlying network caches data), when her device is no longer on the network so only an update of previously known information can be performed. Alg. 3 handles the case when the KNG received from the advert ($gossipKNG_A$) matches the current device knowledge ($localKNG_A$) (derived from previous key server updates, or last confirmed through previous gossip, or both). Note, because localKNG changes with updates from the key server, it means that, provided no equivocation took place, localKNG will always match the signed KNG in the advert. We discuss what happens when equivocation or delays happen and these don't match in Section 3.4.5.

For DIRECTSYNC (Alg. 3 – line 10), the direct connection between devices depends on the underlying network protocol used; we assume that device-to-device secure connections can be established. During this connection, Bob's device checks that it received the same KNG information that was signed in Alice's advert and then proceeds to processing any changes to Alice's KNG (Section 3.4.5).

During SECONDDEGREESYNC (Alg. 3 – line 22), Bob's device verifies the signature using its own knowledge of Alice's KNG ($localKNG_A$): if the verification succeeds and the advert has a newer timestamp than the one held by the device, then Bob's device can refresh Alice's last gossip advert by replacing the previously held advert (signed by Alice) with the new one it just received; but if the signature verification fails, either the KNG in the advert differs from the one held by Bob's device (valid signature on different data) or the signature is corrupt. Either way, when the signature fails and Alice is not on the network, Bob's device cannot update Alice's KNG. If there are multiple occurrences of this event on different adverts and up-to-date information from the key server, then an alert is triggered to inform Bob of the event. We leave investigation into usable ways to present such information to users as future work (Section 6.1.4).

Devices connected via a DIRECTSYNC process can also confirm common friend's adverts by performing a `checkCommonFriends`, which uses private set intersection on the set of names (e.g. phone numbers) to find the friends two users have in common. This extends this one-to-one connection, which is heavily dependent on users co-locating physically with their friends, to a many-to-one by allowing friends-of-friends verifications. Upon obtaining the intersection the two devices exchange the most recent adverts for each of the friends in the intersection and, provided the signatures match the KNG they have on record, they update their friends' KNGs with the most recent gossip advert by running SECONDDEGREESYNC. In particular, if the hashes match and the timestamp is newer than what the device previously held it refreshes with the newer version. If the hashes do not match, we describe the process in Section 3.4.5. Because SECONDDEGREESYNC takes into account the most recent changes confirmed by the server (through localKNG) it ensures the update is still timely albeit, the timestamp could still be very old. Eventually newer signed adverts would propagate to the device, which could confirm the data is still valid, or could enable server misbehaviour detection (server continued to confirm old data despite the device changing this through gossip).

3.4.5 Handling changes to KNGs

Through baseline gossiping, devices automatically check for consistency over time and across users. Although key change is expected to be rare, we nevertheless handle key addition, key removal, and orphaned keys. Tracking these helps distinguish ghost-user attacks from legitimate changes. Because the key server propagates any key changes to a Alice's friends and they update their local view of Alice's KNG, she advertises on local networks only with her newest KNG. This section discusses how we analyse the delta between $localKNG_A$ and $gossipKNG_A$ when a change has occurred.

Key addition. A user logging into the app from a new device (e.g., a desktop or browser version of the mobile app, or adding a new device to a group of devices as in the iMessage model) is a key addition. This can be confirmed by the other devices in the group since new device registration is already a manual process. It often involves users scanning a QR-code from the device on which the user is using already, or using one-time passcodes to confirm the addition from the other devices in the group.

Key removal. A key removal happens when a user signs out of the app on one of their devices. Similar to key addition, this is a manual process that can be confirmed by both the device itself before it signs out and by other devices in the list of user devices.

Orphan keys. An artefact of the gossip protocol is an orphan key. These are created when the last key in the account is removed and replaced but the change cannot be signed and gossiped. These are linked to events in which access to the device (and hence the private key) is lost: device

Algorithm 3 (*Sync*): For every discovered advert in $discoveredAds$, Bob's device runs sync to verify the local KNG against the signed KNG in the advert.

```
1:  procedure SYNC(discoveredAds)
2:      for each $(ad, h) \in discoveredAds$ do
3:          $friendDev \leftarrow precmpHashToDeviceMap.\texttt{get}(h)$
4:          $friendPubKey \leftarrow publicKeys.\texttt{get}(friendDev)$
5:          if canConnect(friendDev) then
6:              return DIRECTSYNC(ad, friendDev, friendPubKey)
7:          else
8:              return SECONDDEGREESYNC (ad, friendPubKey)
9:
10: procedure DIRECTSYNC(ad, frDev, friendPubKey)
11:     if connect$(frDev)$ then
12:         $gossipKNG \leftarrow$ receiveKNG()                          ▷ receive the new KNG
13:         $match \leftarrow$ KNGSMATCH$(ad.sig, friendPubKey, ad.t)$
14:         $localKNG \leftarrow$ getKNG$(friendPubKey)$
15:         $frDev.$PROCESSSYNCDATA$(ad, localKNG, gossipKNG, match)$
16:                     ▷ Update localKNG timestamp, or Alg. 4 if dealing with key change
17:         checkCommonFriends(frDev)
18:                 ▷ runs PSI to enable SECONDDEGREESYNC for common friends' adverts
19:     else
20:         SECONDDEGREESYNC$(ad, friendPubKey)$
21:
22: procedure SECONDDEGREESYNC(ad, pubKey)
23:     if KNGSMATCH$(ad.sig, pubKey, ad.t)$ then
24:         updateFreshness$(ad, pubKey)$
25:                               ▷ Copy the new advert and update KNG timestamp
26:     else
27:         recordFailOrAlert(pubKey)
28:                                 ▷ If threshold is met create a user-facing alert
29:
30: procedure KNGSMATCH(sig, friendPubKey, t)
31:     $hashLocalKNG \leftarrow$ hash(getKNG$(friendPubKey)$)
32:     return verifySig$(sig, friendPubKey, hashLocalKNG, t)$
```

Algorithm 4 (*Process Sync Data*): Process data during DIRECTSYNC. At this stage, Bob's device identifies any key changes that alter Alice's KNG (addition, removal, or orphan keys) by running CALCULATEDELTA between Bob's view of Alice's KNG ($localKNG_A$) and the one received from Alice's device ($gossipKNG_A$), and guided by the SKMs received.

```
1:  procedure PROCESSSYNCDATA(ad, $localKNG_A$, $gossipKNG_A$, match)
2:      if match then                                   ▷ $localKNG_A == gossipKNG_A$
3:          checkSKM()      ▷ check that all entries in $localKNG_A$ received an SKM; update
    SKM list stored locally
4:          $removed \leftarrow localKNG_A$.getRemovedBindings()
5:          checkRemoval($removed, gossipKNG_A.getSKM()$)
6:                                          ▷ check that all previously removed keys have an SKM
7:      else
8:          $(extraLocal, extraRemote) \leftarrow$ CALCULATEDELTA()
9:          reconcileExtras($extraLocal, extraRemote$)
10:                                         ▷ do SKMs account for extras, or still within threshold?
11:
12: procedure CALCULATEDELTA
13:     $extraLocal \leftarrow$ diffList($localKNG_A, gossipKNG_A$)
14:     $extraRemote \leftarrow$ diffList($gossipKNG_A, localKNG_A$)
15:     return($extraLocal, extraRemote$)
```

theft, physical loss or irreversible damage, or by forcibly uninstalling the app and reinstalling on the same device (e.g., for storage management purposes). Loss of access to the private key means the event cannot be confirmed by the device. Typically, when such an event takes place, the user is able to recover their phone number (e.g., through transferring the phone number from the old SIM card to the new, or transferring SIM cards between devices) and re-register for the app. Such changes are accepted by the app after verification: e.g. the app might send a one-time passcode to the user's phone number for them to enter into the app. This replaces the account on the old device with the account on the new device and thus creates an orphan key in the gossip protocol because the new device does not know about any previous devices.

We introduce *signed key modifications (SKMs)* to authenticate key change events. An SKM is a key binding and consists of a timed signature of a hash of a public key (and specifically the key undergoing modification) and an action identifier (the type of modification taking place: key-add or key-remove). The signing key in the SKM belongs to the device signing the change, which can also be the device itself for a removal.

SKMs are used during a DIRECTSYNC when, together with the gossipKNG (Alg. 3 – line 12), the device receives the SKMs for that KNG (including removal SKMs for bindings no longer present). This enables the PROCESSSYNCDATA procedure (Alg. 4) to analyse cryptographic evidence for any changes that took places in Alice's KNG over time and detect malice. We differentiate between key change events using SKMs (Alg. 4 – `checkSKM`, `checkRemoval`, and `reconcileExtras`) as seen in Table 3.1:

We note two special cases: orphaned keys seen gossiping (indicative of a *person-in-the-*

SKM type	$gossipKNG_A$	$localKNG_A$	SKM add	SKM remove
Added & Valid	●	●	✓	–
Removed & Valid	○	○	✓	✓
Added & Pending	●	×	✓	–
Removed & Pending	○	●	✓	✓
Removed & Invalid	●	○	✓	×
Orphan	×	○	✓	×
GhostUser & Inactive	×	○	×	×
GhostUser & Active	×	●	×	–

Table 3.1: The list of key change events that occur within the gossip protocol. We identify active bindings (●), removed bindings (○), missing bindings or SKM (×), received SKM (✓), and not needed or expected SKM (–).

middle attack: the key is still in use but the key server says that it has been replaced), and key change updates arriving out-of-order (local gossip may be faster than a key server update). For the former we record orphan keys and include them in the discover process (Alg. 2), and for the latter we keep two temporary lists of bindings: pending removals and pending additions (Setup), which store cryptographic proof that changes failed to arrive from the key server repeatedly. Key server misbehaviour is observed by a user's friend if the key server fails to acknowledge changes that have already been observed through gossiping in the next key server updates. The number of key server updates chosen for this threshold is configurable.

3.4.6 Proving misbehaviour

The baseline gossip protocol creates evidence for Alice's friends to *detect* key server misbehaviour: they receive Alice's key bindings from the key server, but also the signed KNG and SKMs through gossiping. However, they are unable to prove misbehaviour to an external party (e.g. the app developer or even Alice herself). This is because the key server provides the key bindings to Alice's friend via an authenticated channel, so no signature on the bindings is produced. To prove to a third party that the key server has produced improper key bindings, the key server needs to share cryptographic signatures on the key bindings with users. Hence, when a mismatch is detected, Alice can use the cryptographic evidence acquired to prove to others that an attack took place, but she cannot fake an attack. This is important as faking an attack might undermine confidence in the messaging infrastructure (e.g., by a competitor) and push people to other, possibly less secure, platforms.

3.4.7 Gossiping with strangers

The baseline gossip protocol so far balances mobile device energy consumption and privacy as a means to enable adoption. This is done by limiting gossip to friends co-locating on the local network. We consider two different approaches that relax one of the two restrictions, and discuss the advantages and disadvantages of them.

Friend-of-a-friend gossip enables users to gossip with strangers, if they have a friend in common. This could be enabled by devices initiating private set intersection with strangers to discover if they have any friends in common (i.e. if the stranger is a friend-of-a-friend). Remaining challenges include the high cost of running PSI and the possibility of an attacker running an exhaustive search by having a very large friends list to find out who is friends with whom. These could be mitigated by imposing rate limits and maximum set sizes – a common approach to deter data scraping [35].

To expand the gossip beyond the friendship graph, we could use *epidemic-style gossip*, in which all gossip peers act as data mules. Each gossip peer collects a subset of adverts they see on a network and re-share them on another network. This enables `secondDegreeSync` to happen between friends who would not physically otherwise meet. Remaining challenges include maintaining user privacy and ensuring that local networks do not get flooded with adverts.

3.5 Underlying technologies

Gossip protocols are a class of distributed communication protocols specifically designed for the efficient dissemination of information in large-scale, decentralised systems. These protocols are inspired by the way gossip spreads in social networks and are seen as typically lightweight for the infrastructure needed. In the context of content distribution, gossip protocols can be particularly useful for efficiently disseminating information across a network of nodes. Projects like Haggle [187] and Firechat [24] proposed to use gossip protocols for distributing content between end users, such as files and messages. Notably, Firechat successfully constructed a robust mesh network, empowering smartphones to disseminate messages in a peer-to-peer fashion. It received significant adoption during protests and demonstrations [23], where conventional communication channels are often subject to censorship or face scalability limitations. Matrix [71] provides decentralised encrypted communication by implementing a federated approach to messaging in which the users can subscribe to a static servers (called *homeserver*), that can either be run by the user or someone else. Matrix peer-to-peer [96] was later introduced to test a fully decentralised Matrix protocol in which the homeservers are run in a peer-to-peer/single-user way.

To detect key server equivocation, gossip protocols are a intuitive choice because they provide a lightweight approach to distribute KNGs between friends and ensure that we can tailor the underlying technology to the app requirements. Suitable solutions include Wi-Fi, Bluetooth Low Energy (BLE), the telecommunications network (5G), and peer-to-peer networks. I hosted two interns in 2017 whose projects were designed to implement the gossip protocol steps for both the Android and the Apple platforms. These early prototypes showed that DNS-SD on Wi-Fi is a good fit due to the ability to discover peers on the local network, as well as to leave tokens behind due to it's built-in caching behaviour, despite users having to be connected to the Wi-Fi. Using the IP address and port number fields, devices can establish a secure channel between the two devices to finalise the sync operation. An alternative is the Wi-Fi probing mechanism, which can serve as an overlay for small notifications [3]. BLE is showing promising results in recent deployments: privacy-preserving contact tracing apps for the SARS-CoV2 pandemic [101], as well as the FindMy app, and AirTags on Apple devices. Furthermore, BLE now reaches devices over distances ranging of up to $1\,\mathrm{km}$ using the "coded phy" feature, which is already available in OnePlus phones [4]. Also, 5G device-to-device communication (5G-D2D) [173] specifies that communication between two nearby users can bounce off the local 5G cell tower rather than being routed through the telecommunications network. Lastly, suitable peer-to-peer infrastructures such as Dat [113], Cabal [28], or Matrix P2P [96] reduce the restrictions on physical distance between friends and bring some of the local safety advantages to a global scale, provided that users are connected to each other over a network that has not been tampered with.

3.6 Related work

Public key cryptography enables secure communication over insecure channels, but the end points need to know each other's public keys. PKIs [141] provide the structure needed to bind user identities to public keys. Manual management of keys saw limited deployment due to scalability issues [66, 177]. Other flaws also led to low adoption numbers [63, 76]. Web of Trust (WoT) provides an alternative to traditional PKIs, by decentralising trust establishment. PGP employs the WoT approach, but is famous for its usability issues that lead to ineffective security [192, 228], which in turn limited adoption. Even if all usability issues were addressed, users still face the scalability issue of the manual process. Keybase simplifies PGP for end-users by linking identities to social media [89]. Human interaction has been used previously to create WoT between devices [112]. SMKEX [43] implements a secure opportunistic key exchange, with forward and backward secrecy properties, that leverages the diversity of network paths by creating encrypted tunnels without reliance on network topology.

Automation addresses the scalability and usability issues of the previous approaches, but it assumes an implicit trust, which exposes the end-user to a number of attacks [213, 217]. In secure messaging, manual key verification is proposed as an option to provide transparency, but this results in an error prone process [21, 186, 219]. Some apps provide information on which devices have access to the account, but such notifications may be suppressed in the case of an attack, as suggested by GCHQ [131].

Certificate Transparency (CT) automatically verifies the behaviour of Certificate Authorities (CAs), but it requires CAs to store all the certificates they issue in an append-only log that can be queried publicly and provide proofs of inclusion [127]. Although widely deployed [199], it is at risk of split-view attacks. Solutions to thwart such attacks focus on using gossip protocols [34, 159, 203], but have not been deployed yet [200].

CONIKS [146] proposes using the CT approach for secure messaging key servers, however it has not been deployed due to incompatibility between closed systems such as iMessage or WhatsApp [145]. A CONIKS extension proposes handling multiple devices, but it involves manual pairing [146]. Further improvements on verifiable key directories such as SEEMless [30] and Parakeet [136] optimise the approach proposed by CONIKS to reduce the delay for key updates, as well as reduce the size of the storage needed to record the data structures.

Alternatives propose removing the need of dedicated infrastructure in favour of using blockchain technology [78, 210]. In comparison to previous non-equivocation solutions, our gossip protocol offers a lightweight alternative to discover key server equivocation and detect ghost-user attacks, while providing minimal change to the app server infrastructure in favour of changing the app client to enable gossiping of KNGs.

3.7 Summary

In this chapter, we presented ghost-user attacks, a novel approach for adding covert devices to an individual's account for an end-to-end encrypted messaging app and we proposed a solution to automatically detect such attacks by using gossiping. We started by formalising the key-to-name bindings used by key servers in popular secure messaging apps into key-name graphs. This structure is useful to model any combination of key-to-name bindings the apps use and is used by all the devices during the gossip process. We also formalised the gossip protocol steps to reveal how gossiping between the devices of friends can take place. Handling the key changes that can affect a key-name graph is an essential step in the detection of ghost-user attacks because part of the subtlety of the attack is not having any mechanism to differentiate between legitimate key changes and malicious ones. We also take into account the necessity for a way to prove to third parties when an attack is discovered, and we detail the changes required at the key server level to enable such abilities. Gossiping more widely within the friendship group, such as with friends-of-friends, or even strangers is something we consider as an enhancement to the baseline gossip protocol in this chapter. We also discussed the multiple options for the underlying network technologies, which can be tailored depending on app choice, location or other factors. We conclude the chapter with a related work review. The next chapter covers the analysis of the baseline gossip protocol, by implementing the algorithms presented for the baseline protocol in this chapter into a discrete event simulator. The simulator helps us analyse the protocol's main security goals 6.1.3. We also look at two different data sets that help us analyse the protocol's effectiveness.

CHAPTER 4

BASELINE GOSSIP PROTOCOL ANALYSIS

In Chapter 3, we described ghost-user attacks, a subtle targeted wiretapping approach aimed at end-to-end encrypted messaging users, and proposed a baseline gossip protocol to enable end-user devices to detect when the key server equivocated, and, as a consequence, identify when a ghost-user attack or a person-in-the-middle attack has taken place. The baseline gossip protocol is supported in Chapter 3 by several enhancement proposals to enable gossiping with strangers and proving misbehaviour. In this chapter, we analyse the baseline gossip protocol and justify it meets the security goals defined in Section 6.1.3. We also analyse the protocol's effectiveness by using two datasets: Wi-Fi connectivity from 11 485 Android devices between 2011-2016, as well as call-detail records data coming from over 1 million devices in India for a one-month period in 2012.

My contribution to this work has been to perform both the effectiveness and security analysis. I implemented the Java-based simulator, which includes the gossip protocol steps, as well as adapting the simulator to support model checking. For the call-detail records analysis I went as a visiting researcher for four months in 2019 at the company hosting the dataset in order to perform the analysis on their servers (subject to an NDA and ethics approval by the university's Department of Computer Science and Technology). Throughout this work and the writing up of the results I was advised by both Daniel and Alastair who provided invaluable advice on both the analyses and the write-up. The prototype implementations (Section 4.1.3) in Android and iOS were implemented by two summer interns in 2016 under my guidance, whom I've jointly hosted with Alastair.

To summarise, the main contributions of this chapter are:

- a simulator implementation to check the protocol steps and subsequent analysis through model checking (Section 4.1.1);
- an informal security and privacy analysis focused on the core security goals of the baseline gossip protocol (6.1.3), using data from the simulator and model checker (Section 4.1.4);

and

- an effectiveness analysis using using Wi-Fi usage data from 11 485 Android devices to quantify the effectiveness of a local Wi-Fi gossip protocol and the social-geographic analysis of over 67 million call-detail records belonging to over 1 million GSM subscribers, over a 35-day period (Section 4.2.3).

4.1 Privacy and Security

We begin by describing our implementation of the simulation platform, which includes a Java implementation of the algorithms in Section 3.4. We also detail the changes we made to turn the simulator into a model checker. We discuss early prototype apps that tested the use of mDNS as an underlying network technology to perform gossiping. We use the simulator and model checker to justify how the baseline gossip protocol meets its core security goals (Section 6.1.3).

4.1.1 Simulation

We implemented the baseline gossip protocol as described in Section 3.4 in Java and exercised it in a discrete event simulator. We abstract network details and leave systems-style analysis, such as performance and bandwidth for future work. We use the simulator to test the gossip process, and to provide substance to our security analysis.

In the simulator, we identify each person by a unique name. Each person can have one or more devices. The first device that is associated with a person is marked as the primary device. If the primary device is removed, one of the remaining devices deterministically becomes the primary device. When creating the person in the simulator and adding a first device, we register an account with the key server using the triple (name, device public key, device ID). The user can add or remove key bindings from the account, but if the last remaining device is removed, the account gets removed from the key server. The key server also provides account updates to any device that enquires about a name.

Devices hold a list of friends (to represent an address book on a smartphone). Together with the list of friends, the device also maintains a list of KNGs belonging to the friends' devices, as well as a KNG for the user's own account. For each KNG, there is a list of pending additions, one for pending removals, one for removed key bindings, and the actual key name graph, which stores the active bindings, as guided by the protocol specifications in Section 3.4.4. Alongside each of the entries in these lists, we record a list for the SKMs received from gossiping.

Every time a user adds a new device, each one of the user's devices (including the new one) sign this change and record the SKM for it in the user's KNG for later gossiping. Also, every time a user removes a device, all of the devices sign this removal and it is recorded in the user's removed key bindings list in the KNG. These signed modifications help with conflict resolution during the sync step of the gossip protocol.

Each time-step in the simulator sees events happening in the same order: *movement* (processing of any key addition/removal, devices coming/leaving locations, ghost keys addition/removal, friend addition/removal), *key server updates* (every device receives updates from the key server for every friend in its list), *advertise* (all the users who come to a location, or whose adverts have expired at the location they are in currently, prepare a fresh advert), *discover* (all the devices present in locations run a discover step in which they find out which of the currently present adverts belong to their friends), *sync* (all of the devices that have non-empty lists of discovered adverts attempt to perform a sync with the devices that advertised), and *stats* gathering.

The simulator takes as input a Json config file specifying the users in the simulation, their friendship network, and movement over time. For each user, we specify in the config how many devices they have. The simulator then generates public-private key pairs for each of them, adding them to the user's KNG and notifying the key server.

We used unit testing and more explicit gossip step testing to check that the basics of the protocol result in the expected behaviour. This helped us check the interpretation of key changes in the KNG and how that affects the KNG representation and, most importantly, that the different key changes that can occur on an account are identified correctly.

4.1.2 Model checking

We adapted the simulator to support bounded probabilistic model checking. We create a new config generator, which takes as input the number of people and number of time-steps and generates all the possible combinations of friendship these could have and combinations of steps these people would perform. We also define a *step tree* to represent the tree formed of all the possible combinations of actions a device can perform in the gossip protocol: arrive at a local network, leave a local network, add device, remove device, add friend, remove friend, add ghost and remove ghost. The model checker allows us to perform a stochastic exploration of the step tree. Due to the nature of the protocol this is a parametric model checking problem, for example it depends on the number of users, the maximum friends, the number of devices, and the combination of possible events. The problem is undecidable without bounds set for the model checking to be feasible. We built a list of all possible friendship combinations up to 5 users (14 700 combinations). We kept the same number of devices per user and same order of the events (step tree) across the friendship combinations in these simulations. We generated config files under these starting restrictions to enable us to perform a stochastic exploration of the step tree.

4.1.3 Prototype implementations

We implemented basic prototypes of the gossip protocol for both the Android and iOS platforms that use the Bonjour protocol [51] to perform the advertise-discover process, which allows users to find each other in a zero-configuration approach on the local network. The main result of the prototype was to demonstrate that zero-configuration protocols work between Android and iOS devices and can therefore serve as the basis for a practical implementation over Wi-Fi.

4.1.4 Privacy and Security Analysis

We analyse the gossip protocol's core security properties.

Correctness: the protocol correctly identifies all the types of key changes for KNGs (Section 3.4.5). Unit testing and gossip step testing checked that the basics of the protocol result in the expected behaviour. We verified the interpretation of key changes in the KNG and that every possible key change is identified correctly. This identified one case in which a false positive is returned: the user forcibly replaces their device before gossiping, resulting in an orphaned key. This is because all of the user's friends' devices will have recorded the addition and then removal of the key for the removed device and would be awaiting confirmation through an SKM, but the user's new device doesn't know of the key. This is expected to be a rare event and something that the end-user will know has happened. Without modifications to the key server this can be addressed with user-facing prompts (e.g., "Did you recently change devices?"). Alternatively, the key server could store and forward SKMs in each account: the user's device confirms key modifications through SKMs. These SKMs are delivered during user updates, which leads to identifying the orphaned key correctly: it will not have a removal SKM, but it will have the addition SKM, which is essential for differentiating between legitimate key additions and ghost-user key additions. However, server-side changes are out of scope for this paper.

Soundness: the gossip protocol eventually detects ghost-user attacks when one has taken place, provided the user successfully gossips with at least one friend. Model checking shows that this happens even if the ghost has been removed by the attacker because there would still be a record of the addition and then removal of the key on the user's friends' devices and no confirmation from the user's device.

Availability: if the ability to gossip is enabled, users *can* gossip. An attacker can cause denial of service on a particular local network by flooding many spurious gossip messages so that there is either no more space to store gossip messages on the network or the devices give up checking hashes before they come to a real message. However, local network denial of service is fairly straightforward without gossiping and as long as there is at least one local network not affected by an attacker then gossiping can succeed on that network. Targeted denial of service attacks are also possible. An attacker can overwrite adverts for a user or repeatedly attempt to gossip with a device until it hits a rate limit or runs out of power. The latter can be mitigated by targeted rate limiting of gossip attempts, but this can be difficult if the attacker changes their MAC. However, just sending a high volume of IP packets to the targeted device is likely to have a similar impact.

Privacy: the protocol aims to maintain the privacy protections offered by the app, while improving the security of connections between end-users. It does not share persistent identifiers on the network, because $H(H(K), T)$ changes with each timestamp and can only be decoded by those who already know K. Similarly, the signature of the KNG $\{H(G), T\}_{Sig_{K-1}}$ is opaque such that only those who know the graph itself and K can verify it. Hence, anyone with access to the user's public keys can detect the device's presence on the network. Without collusion with the key server, only users' contacts that are physically close to them can detect their (past) presence on local networks. If a nation-state attacker colluded with the key server, they would have the power to pre-compute the KNGs for all the users to test for the user's presence on the local networks. However, on modern Wi-Fi networks, connected devices already use the MAC address of the Wi-Fi chipset, which can act as a static identifier despite randomisation attempts [69, 138, 140]. Such information is already trivial to gather by the owner of the access point, so it would more sensible just to use the MAC address for such tracking. However, to mitigate the risk of a nation-state attacker colluding with the key server, additional changes at the key server level, or more reliance on zero-knowledge algorithms is required, which we leave as further work. Gossiping may become identifying if only a small proportion of devices gossip. Furthermore, PSI gossiping (Section 3.4.7) leaks information about any shared contacts during a set comparison with another device. If an attacker has the public key of a user and inflates their own contact list to check for contacts of interest (e.g. known dissidents) they can quickly confirm if the victim has that contact. This risk can be mitigated by enforcing a maximum contact list size, rate limiting gossiping, and specifying contacts to hide from the list at least for some contacts.

4.2 Effectiveness

Due to the restrictions imposed by the baseline gossip protocol so that only users who know each other can gossip, it is important to analyse the effectiveness in practice of such an approach. To perform this analysis we use two different data sets: the Device Analyzer dataset [221], including data collected from 11 485 Android devices from 2011–2016, and a call-detail records (*CDR*) dataset from over 1 million subscribers generating over 67 million records in a 35-day period in 2012.

4.2.1 Device Analyzer

Device Analyzer (*DA*) [221] is the largest capture of the Android ecosystem in the wild. It was a project run by researchers at the University of Cambridge for over a decade (2010 – 2020) to capture usage information directly from Android devices. The project continuously collected Android device usage data in the background directly from Android phones through a purposely built app. Users can install the app from the Google Play Store and need to agree to participate in the study in order for the phone to send the stats for collection. The app also displays summary statistics about the individual device for the users of that device to browse. DA records more than 200 different usage statistics from the Android device, from raw telephony and sensor data, to device and app usage [10].

In DA, direct personal identifiers are removed before leaving the end device. Data entries can be correlated between re-installs of DA on the same device, because DA's data has a salted hash function applied on the data, which is derived from on-device hardware identifiers, but it cannot be correlated across devices. Furthermore, multiple options are presented to users to provide enhanced user ownership over their data, such as downloading their historic raw data from the DA website based on their unique identifier, choosing whether the collected data gets sent for research or discarded, opting to pause or withdraw from data collection at any point in time, and even asking to have their historic data deleted [221].

Our research is based on a snapshot of the DA dataset taken in 2016. The dataset contains over 18TB of exported raw historical data over six years (2011 – 2016), collected in 30 445 compressed directories containing daily log files. Contribution times vary significantly, with some devices contributing data for less than seven days and some more than a year. Because we are looking for longitudinal patterns in the data, we eliminated from our analysis those devices which contributed data for less than 30 days. This restriction reduces the sample dataset from 30 445 to 11 485 devices in total worldwide.

Limitations

The main limitation for using DA in our effectiveness analysis is the inability to correlate information across multiple devices. For instance, we cannot extract co-location information because, even if a Wi-Fi access point was shared between multiple devices, these would appear as different entries since the hashing function is device-dependent. Furthermore, the privacy enhancements mean that no demographic or biographical data is available either. Lastly, device contributions are not always complete due to app crashes or connectivity issues, or consistent in terms of length of contribution. We chose a minimum contribution length of 7 days, which resulted in a 62% reduction in devices considered for the dataset (from over 30 445 contributing devices we reduced the set to 11 485 devices, which contributed data for at least 7 days).

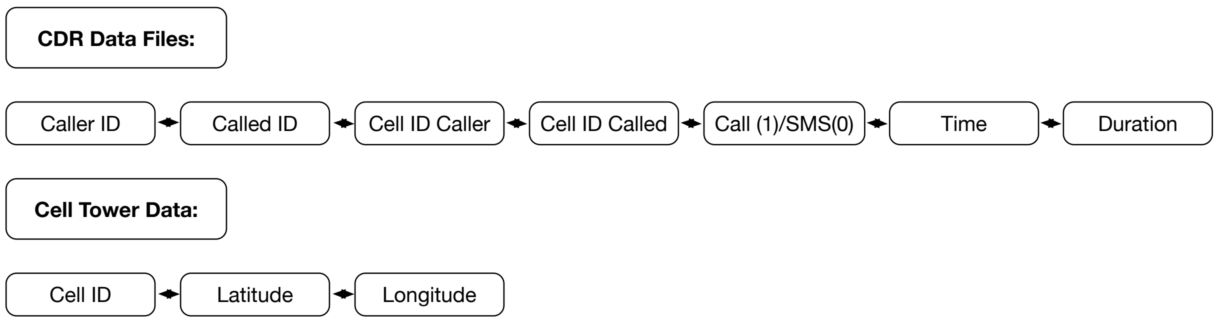


Figure 4.1: The dataset format after the pre-processing step of anonymisation

Ethical considerations

We obtained permission prior to conducting our analysis on the 2016 snapshot from the DA dataset from the University of Cambridge Department of Computer Science and Technology's research ethics committee.

4.2.2 Call detail records

We analyse a CDR dataset consisting of over 67 million CDRs generated by more than one million subscribers over a period of 35 days in the spring of 2012 in Karnataka, one of the largest regions in India.

The data comes from all active mobile subscribers of a telecommunications provider in Karnataka. Karnataka is a large region with a surface area that covers over 5% of India's total land mass. It consists largely of rural areas, with some larger cities along the coastline and the region capital, Bangalore, which has a large surface area of over $700\,\mathrm{km}^2$. The cell provider had over 5.7% of the total GSM subscribers in the region at the end of April 2012 [163].

The dataset largely consists of two types of files as seen in Figure 4.1. There are files for each of the 35 days filled with rows of CDRs, consisting of subscriber ID, phone number, cell tower ID for both caller and called, as well as time of the call, whether it was a call or an SMS and duration. The dataset is also accompanied by a file containing cell tower IDs with latitude and longitude coordinates. There are three consecutive cell tower IDs for every one of the cell tower's antennae, as explained in Section 4.2.3.

Cell tower IDs were pseudonymised before we received the data so cannot be directly linked to other datasets. We pseudonymised the user identifiers before processing the dataset by applying a salted hash over the subscriber ID and phone number and afterwards discarding the salt, which resulted in the modified records seen in Figure 4.1. We only processed the pseudonymised version of the data for our analysis. All of the computation and analysis was performed by me on the servers of the company hosting the dataset. While it has been shown that pseudonymised CDR data still leaks valuable information that could lead to re-identification of individuals [233], we limited our analysis to the extraction and presentation of non-identifiable (typically aggregated) information while working on the data; we provide further details in our discussion of the ethics approval at the end of Section 4.2.2.

The dataset is a real-world portrayal of communications in the Karnataka region in India for one provider. We thus considered different types of data sanitisation and validation in order to ensure there were no spurious records or inconsistencies. As part of the data sanitisation process, we ensured all entries were semantically valid. We also considered filtering data devices, which might represent shops or businesses rather than personal devices, however there is no clear divide in the data. For example, we attempted to remove subscribers with a large number of CDR entries who do not change cell for the duration of the dataset; however, no such subscribers exist and we have no ground truth data to help us check the validity of more sophisticated data sanitisation techniques. We therefore present our results using data from the full range of subscribers.

We also looked at all cell towers for which we have location information to ensure they were in service and find that each cell tower appears in at least two CDR records. We do have CDR entries containing cell towers for which we do not have location information, which suggests they are either from outside the region of study (but from the same provider), or from a different provider. We decided to retain data collected from these cell towers as this type of interaction allows us to estimate friendship connections and co-location which would otherwise be missing, as outlined in Figure 4.2.

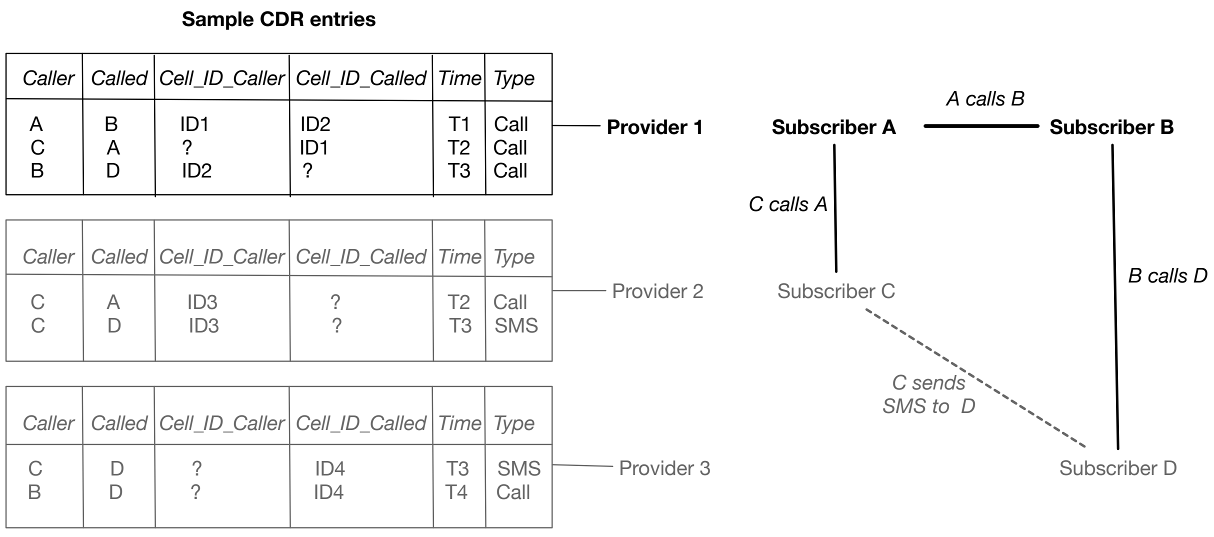
Limitations

While this dataset is useful, there are several limitations.

First, the data comes from a single region in India, Karnataka in 2012 and so the results may not be generalisable to other locations or time period. We argue it provides a lower bound as connectivity is likely to be better both in Karnataka in 2020 and in many locations elsewhere. Furthermore, the main advantage of working with CDR data from 2012 is that it is likely to have better friendship information than more recent data due to the limited alternatives for communication at the time; newer CDR datasets will lack social network information due to the replacement of SMS and voice calls by Internet-enabled applications, such as WhatsApp.

Second, each cell provider is only responsible for monitoring and recording CDR data for its own subscribers and thus the dataset does not record details concerning subscribers of other providers. We only receive data from subscribers of other networks when at least one of the parties to a call or text message is connected to the provider's network. In such cases the provider can record the location, phone number and subscriber ID of the other party, for instance subscriber C and D in Figure 4.2). However, the location for subscribers outside of our provider's network is opaque as the cell tower IDs are pseudonymised and we do not know their location.

Lastly, the location inference is coarse and sporadic because the cell tower distances vary significantly and we only receive localisation information when a two parties communicate via call or text message. The installation of GSM antennas is coverage bound in rural areas, and



Caller	Called	Cell_ID_Caller	Cell_ID_Called	Time	Type
A	B	ID1	ID2	T1	Call
C	A	?	ID1	T2	Call
B	D	ID2	?	T3	Call

Caller	Called	Cell_ID_Caller	Cell_ID_Called	Time	Type
C	A	ID3	?	T2	Call
C	D	ID3	?	T3	SMS

Caller	Called	Cell_ID_Caller	Cell_ID_Called	Time	Type
C	D	?	ID4	T3	SMS
B	D	?	ID4	T4	Call

Figure 4.2: Four GSM subscribers from three different telecommunications providers and some sample interactions between the subscribers with CDR entries linked to each provider; Provider 1 can only record information from Subscriber A and Subscriber B, but will only capture details about Subscriber C and Subscriber D if they are directly involved in a call transaction with one of Provider 1's subscribers.

capacity-oriented in urban areas; consequently location accuracy varies across regions.

Ethical considerations

We obtained permission to conduct this study from the University of Cambridge Department of Computer Science and Technology's research ethics committee. The ethics committee approved the review request with a waiver for informed consent guided by the requirements set out in Section C.2.1 of the Menlo Report [161], such that: *a)* The research involves no more than minimal risk to the subjects; *b)* The waiver or alteration will not adversely affect the rights and welfare of the subjects; *c)* The research could not practicably be carried out without the waiver or alteration; and *d)* Whenever appropriate, the subjects will be provided with additional pertinent information after participation.

We satisfied these requirements as follows: *A)* The research was carried out on the premises of the company that provided access to the dataset and a non-disclosure agreement was signed. A pre-processing step was applied to hash any personally identifiable information in the dataset so this information was not visible to researchers; *B)* the research carried out only generates non-identifying statistics to answer our research questions as set out in the ethics application and no individuals involved in the data will be identified; *C)* Obtaining informed consent from over a million participants is infeasible, especially given the age of the dataset and the fact that the cell provider no longer operates; furthermore, we would be unable to answer our research questions unless we looked at the population dynamics over a large population and geographical area; and *D)* upon publication of the research we will write a blog post presenting our results to the general public; we will happily answer questions from both the general public and any potential study participants.

4.2.3 Effectiveness Analysis

The ubiquity of mobile devices and local networks suggests that gossip protocols are feasible, however our privacy-enhancements of the baseline gossip protocol reduce the opportunity to gossip because it can only take place if users are friends. Our approach thus depends on human mobility in the physical world, social behaviour in the online world, and the availability of local networks to bridge these two worlds.

We start by reviewing friendship models, particularly looking at research on the cognitive limitation of the human brain to maintain more than 150 friendships at a time, and how this is reflected in online behaviour. We then use the two datasets described in Section 4.2.1 and Section 4.2.2 to confirm our hypotheses that local networks are suitable for this approach and that users would physically co-locate with friends on local networks.

Friendship models

Evolutionary psychology shows that people actively maintain at most around 150 friendships at a time [60], commonly known as *the Dunbar number*, because of the biological limitation imposed by the size of the brain's neocortex. The Dunbar number consists of people with whom we have meaningful interactions. Analysis on datasets built from answers to surveys [61], as well as data from Facebook social graphs [15, 62] and Twitter messages [15, 62, 85] have also shown the Dunbar number reflected in online interaction.

The baseline gossip protocol aims to protect the connection a user has with their closest connections, i.e., those whom the user communicates with regularly. Based on the Dunbar number [60], we therefore expect the number of friends a gossip protocol might plausibly protect to be in the range of 15 to 50, with an upper bound of 150. In our experiments on the Device Analyzer dataset, we analysed contacts list sizes from a subset of 7 051 devices that share internal contacts list data with Device Analyzer. These devices had a median of 175 contacts in their contacts list and a 95th percentile of 949. Our analysis of the CDR dataset shows similar results: over 99.88% of the subscribers have 50 or fewer contacts with whom they call or message over the data collection period, and the majority of subscribers actually have fewer than 15 friends. Furthermore, users in the CDR dataset have significantly more friends-of-friends than friends (Figure 4.3). On average, subscribers have a mean of 3.5 direct friends (median 2) and a mean for friends-of-friends of 22.6 (median 10).

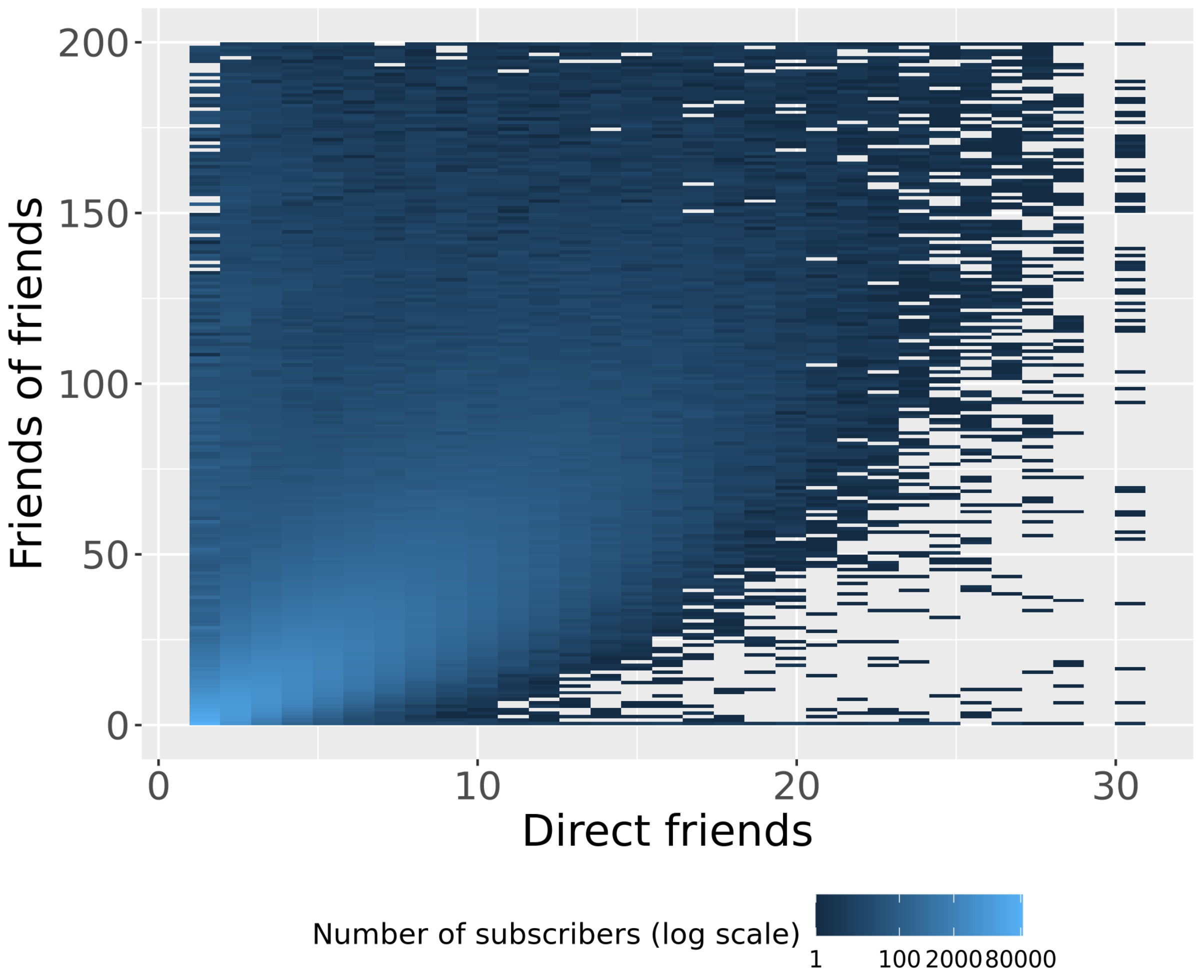


Figure 4.3: Friends and friends-of-friends for each subscriber has in the CDR dataset. There is a higher mean for friends-of-friends per subscriber. Also, a proportion of 2.6% of subscribers have zero friends-of-friends in the dataset due to them either residing outside of the study area, or having a different provider (Section 4.2.2). Plot area is truncated. Max direct friends 910, max friends of friends 15 556.

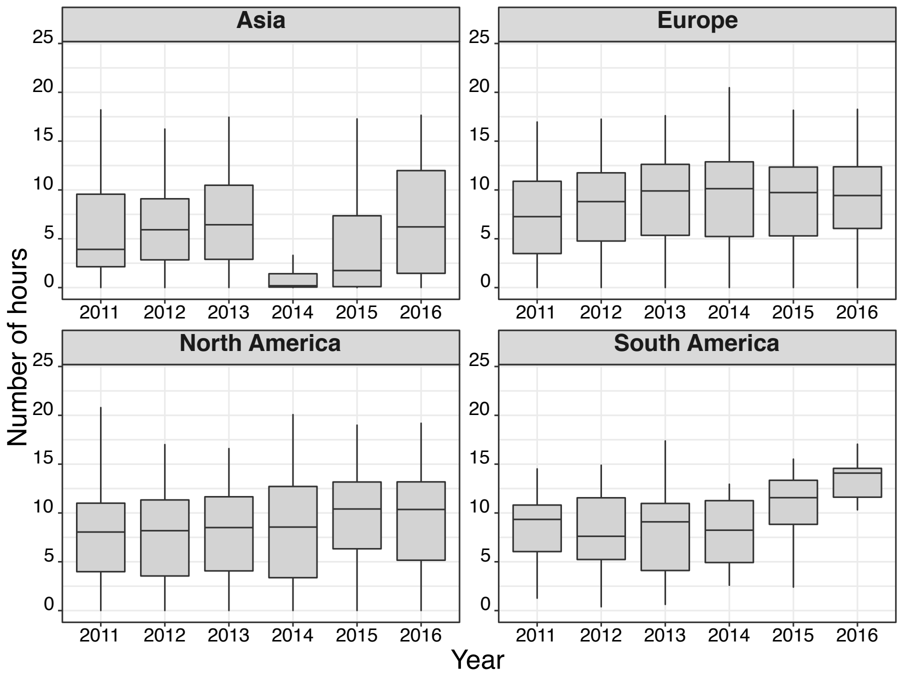


Figure 4.4: Number of hours-per-day spent connected to a local Wi-Fi network for four continents. We had insufficient data to plot meaningful results for Africa. The connectivity data in Asia changed markedly in 2014 and 2015 due to a local study causing a spike in Device Analyzer installations in Bangladesh.

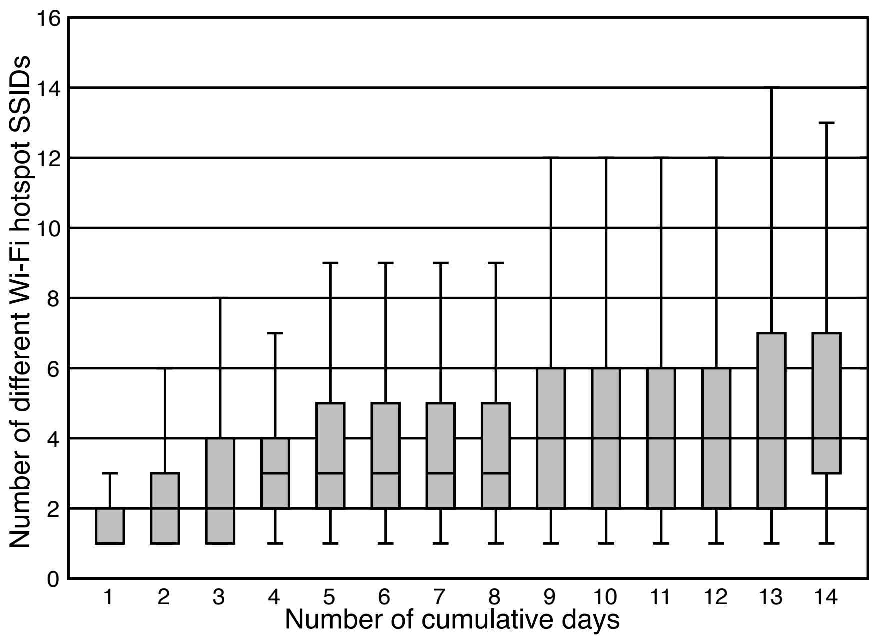


Figure 4.5: Distribution of the number local Wi-Fi networks to which users connected, grouped by the total number of consecutive days considered in the analysis. For example, if the cumulative number of days is four, then we calculate the distribution of the number of local Wi-Fi networks each device connects to across all disjoint four-day periods from all devices in the dataset.

Number of	Days user connected to the same network						
networks	1	2	3	4	5	6	7
1	94.0	81.0	72.7	66.7	61.4	55.9	44.5
2	80.5	71.9	64.1	58.1	51.4	43.5	31.4
3	73.3	63.4	56.2	49.7	44.2	35.7	24.7
4	67.5	58.3	50.6	44.6	39.2	32.3	20.5
5	63.9	53.3	45.8	39.7	33.0	25.9	15.7
6	58.5	48.8	43.3	36.7	31.7	22.9	12.3
7	54.4	45.3	37.2	32.2	26.6	18.8	9.8

Table 4.1: Percentage of Device Analyzer users who connected to at least one, and up to seven, local Wi-Fi networks every day for between one and seven days in total. Data comes from all 733 Device Analyzer users who contributed between 2015 and 2016. Data from North America and Europe look similar.

Wi-Fi Usage Data

We use the DA dataset to explore our hypothesis that local Wi-Fi networks are suitable for using as an underlying network for our baseline gossip protocol. We thus analyse the availability of local Wi-Fi networks.

For each device in our Device Analyzer dataset, we extracted the following information: date, time, duration of connection to a local Wi-Fi network, and the salted hash of the local Wi-Fi network name (SSID). The latter allows us to determine if and when a device connects to the same local Wi-Fi network, but not to correlate connections between devices. If the user of the device has enabled location services and agreed to share their data with us, the dataset also contains the network-based location of the device, expressed as a latitude and longitude. We used the location data to determine the continent of the device, in order to account for different usage patterns.

We observe that devices spend long periods of time connected to local Wi-Fi networks (Figure 4.4): the average device spends around 10 hours a day connected to Wi-Fi networks, with the median times varying between 6 and 14 hours per day on four different continents. There is a significant increase in the lengths of time spent connected to Wi-Fi networks across all four continents, which suggest increasing Wi-Fi coverage over the measurement period.

Furthermore, there is availability of networks to enable devices to display habitual behaviour over 14-day periods (Fig. 4.5): after five consecutive days, the diversity of connections to Wi-Fi access points starts to plateau and after nine days, little additional change occurs for many devices. This is because humans are habitual. We revisit the same place frequently, and seek out new places (and therefore new local Wi-Fi networks) relatively rarely.

Lastly, because trends show that connectivity times have increased over the years (Figure 4.4), we reduced the set size to users who actively contributed in 2015 and 2016 to take a closer look. We split each individual contribution into seven-day periods, which were then used to capture

	Mean	Median	Lower 10%	Top 90%
Within city	12	6.8	1.8	22
Elsewhere only	138	32.5	2.9	299
Elsewhere to city	269	122	17	499
City to elsewhere	273	118	17	502
Full dataset	106	13.4	2.4	250

Table 4.2: Physical distance (km) between the cell towers for people who are actively communicating.

the number of connections to, and diversity of, local Wi-Fi networks (Table 4.1). We observe that a significant proportion of these users (94%) connect to one network at least one day of the week in all of the weeks they contributed to the dataset. More importantly though, over 50.6% of DA devices connected to 4 networks for 3 consecutive days, showing a tendency for devices to reliably connect to some networks over several days (i.e., there are other repetitive connections beside the home Wi-Fi, which is probably the connection that appears most often).

3G Location Data

Location data can be used to infer some properties of the social behaviour of users, however, this requires the ability to correlate location between individuals. Common indicators of friendship include phone communication, their unique locations, and co-location on a Saturday night [65]. Virtual friends are often physically co-located during the day with as many as 10% of Facebook friends co-locating regularly [45]. While the DA dataset is useful for studying individual mobile device connectivity at scale, the study was not designed to capture interactions between different devices, so we looked at an alternative dataset that can provide this element of social interaction, while still capturing relative geographical location over time. Thus, we analyse the social-geographic-temporal aspect of CDR data (Section 4.2.2).

Location in the CDR dataset is given by the cell tower ID entry in the record of a communicating party (regardless of whether they sent or received a call or an SMS). Our dataset contains over 2 000 cell towers with latitude and longitude coordinates, which allows us to observe the type of coverage these exhibit. A GSM cell tower has three directional antennas, each covering a 120° angle, which is referred to as a *sector*. All the three antennas create a *cell*. We therefore define co-location of two users as both devices having the same cell tower ID in the CDR at the time of the CDR being recorded.

In this dataset the distance between cell towers acts as a proxy for the physical distance between subscribers in the CDR data (Table 4.2). GSM cell towers have a maximum range of 35 km. However, we see a higher density of cell towers in cities and along the coastline, which is due to a higher population density and, hence, higher communication demand (Figure 4.6). We choose the largest city in the dataset, Bangalore (Figure 4.7), to obtain a more accurate location of users since each cell tower ID has a smaller coverage area. On the other hand, the large

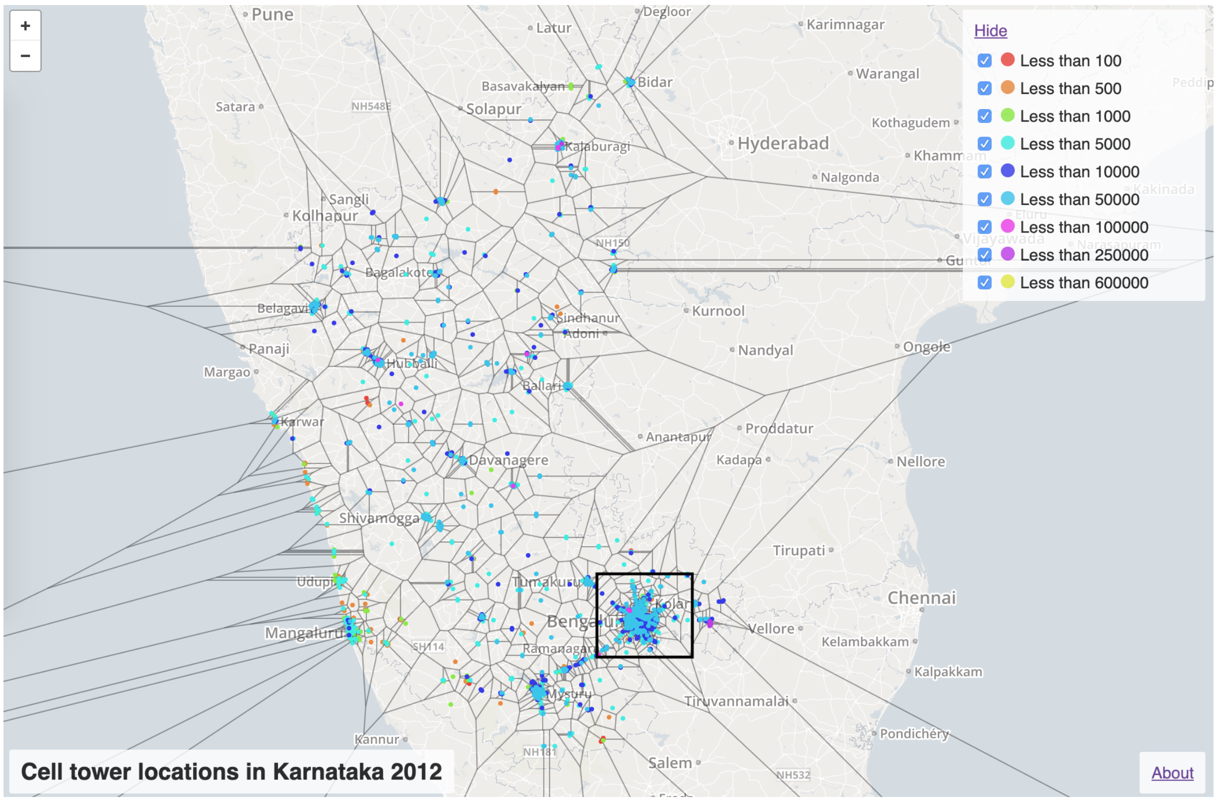


Figure 4.6: The Voronoi tessellation of the cell towers within the Karnataka region; there are over 2 000 cell towers within the full area. It is clear that there is a higher concentration of cell towers across the coastline and in the cities. The square identifies the highest concentration of cell towers, located in Bangalore.

physical distance between the chosen city and elsewhere in the region, coupled with large areas of no connectivity, drive the overall mean distance between subscribers to $106\,\mathrm{km}$. However, these events only represent 16.2% of the CDRs and thus the median is much smaller ($13.4\,\mathrm{km}$).

Within the city, there is approximately 1 co-location for every 5 non-co-located calls, whereas elsewhere in the region this increases to 1 co-location event for every 3 non-co-located calls (Table 4.3). The distances between cell towers and the coverage of each cell tower suggests that on average, when actively communicating, 1 in 5 friends are located within, at most, $4.5\,\mathrm{km}$ of each other within the city, and 1 in 3 friends are within, at most, $17\,\mathrm{km}$ (Table 4.4). The lower number of co-locations within the city is due to the higher density of cell-towers, resulting in smaller coverage areas (Table 4.4).

One of the most significant results shows that over 52.7% of the 1 million subscribers makes at least one sector co-located call during the 35 days. Furthermore, this increases to 58.2% of the dataset population if we treat the cell tower area as a whole (i.e., across all three sectors) rather than look at each sector individually. However, this only captures co-locations that occurred when friends were calling each other but not co-locations that we can infer from calls with others.

We therefore extend our analysis to co-location between friends who are not actively communicating with each other, but are communicating with other parties from the same cell sector

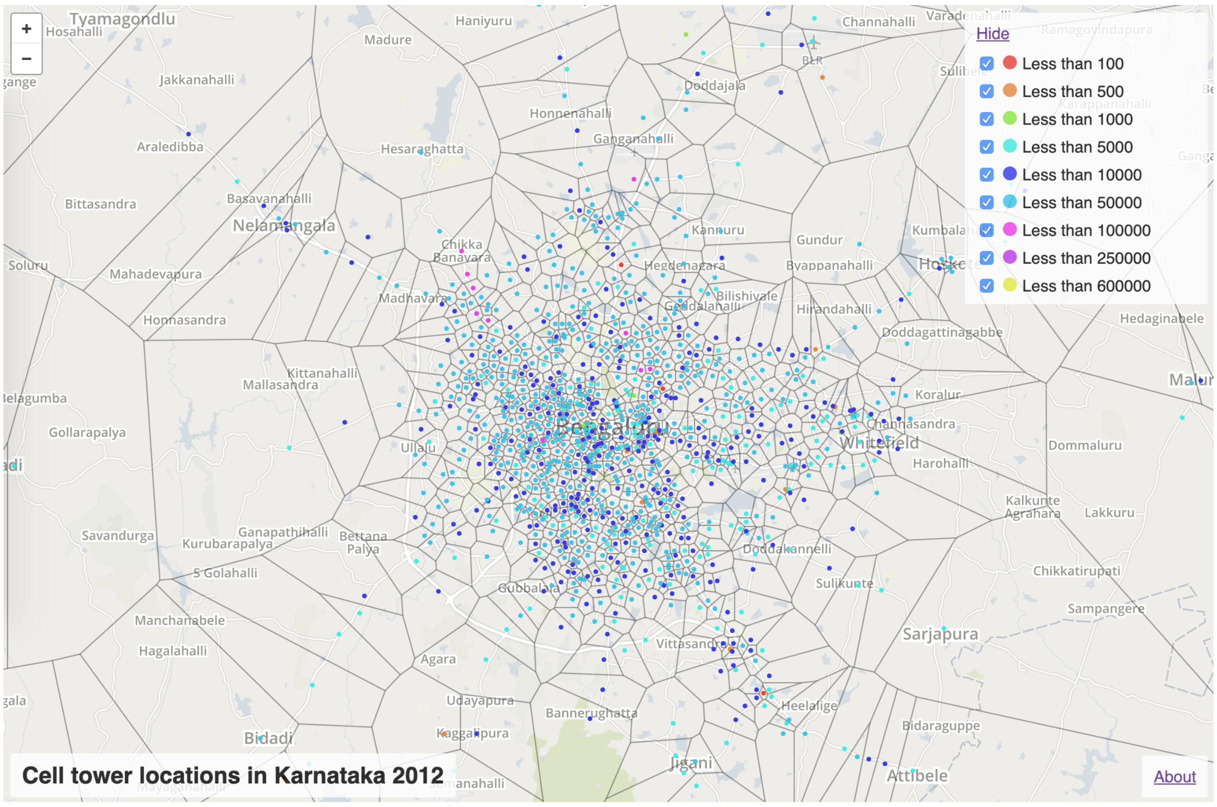


Figure 4.7: The Voronoi tessellation of the cell towers within Bangalore, the largest city from the Karnataka region; there are over 900 cell towers within the city limits

during the same 10-minute interval. We analyse the frequency of subscribers who co-locate with their friends, be they direct friends or friends of friends (Figure 4.8). We find that over 70% of subscribers have co-located with at least one friend (a contact with whom they actively communicated) within the same sector and that all subscribers have co-located with at least one friend-of-a-friend in a cell within the 35 days observation period.

4.3 Summary

In this chapter we reviewed two datasets to gain insight into the effectiveness of the gossip-based key server equivocation detection protocol, which was presented in Chapter 3. We also present the Java implementation of the gossip protocol steps in a discrete-event simulator, which we used as a basis for a security analysis of the baseline gossip protocol (specification in Section 3.4). We conclude that using gossip, an approach which has seen recent widespread deployment for devices in physical proximity to share small encrypted tokes, is a valid solution to enable the detection of ghost-user attacks and person-in-the-middle attacks within end-to-end encrypted messaging apps.

End-to-end encryption has been deployed to the masses, enabling the average user to communicate securely with their friends. But this is not enough in select user groups, such as

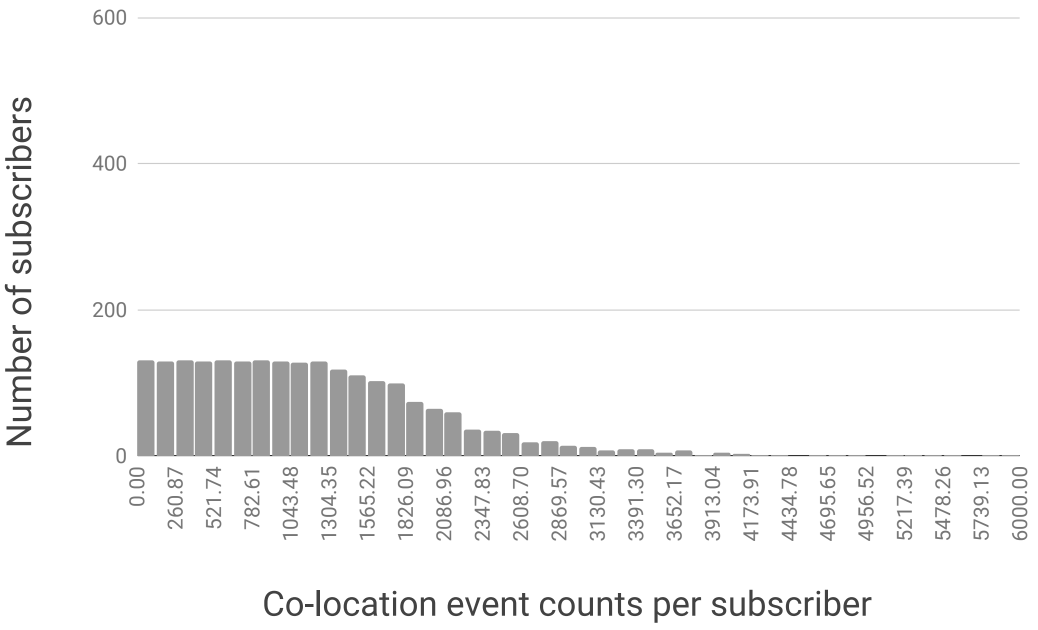
	Co-located	Not Co-located
Elsewhere only	13.4%	35.4%
Within city	6.3%	32.1%
Elsewhere to city	–	6.6%
City to elsewhere	–	6.2%

Table 4.3: Percentage of co-location events seen in the CDRs, excluding over 0.4% of the CDR data that contains at least one cell tower ID we don't know. The "Elsewhere only" notation means everywhere in the region excluding the city area.

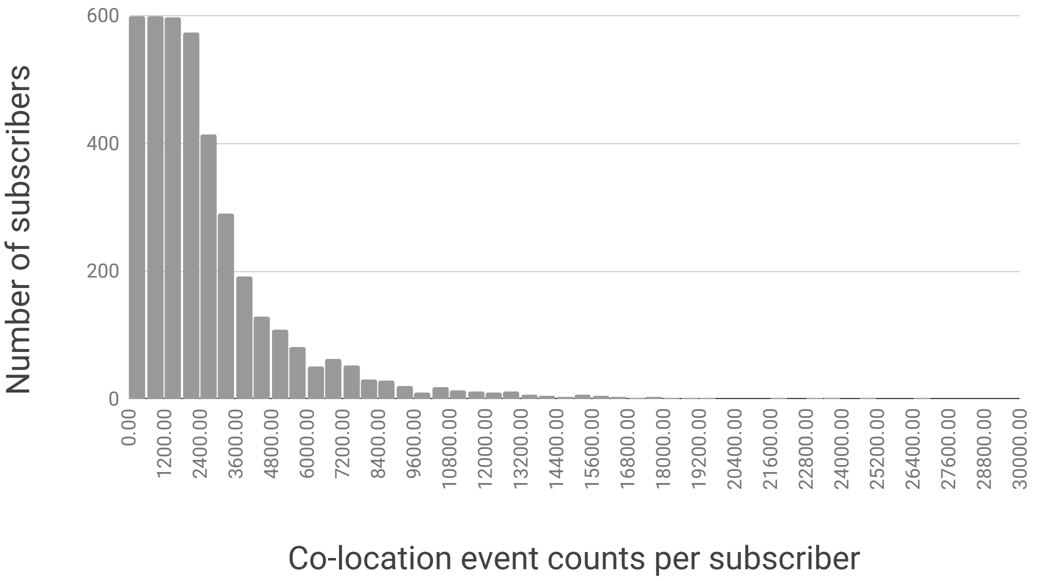
	Mean		Median	
	Area	Range	Area	Range
Region	$925.2\,\mathrm{km}^2$	$17\,\mathrm{km}$	$37.3\,\mathrm{km}^2$	$3.5\,\mathrm{km}$
City	$62.5\,\mathrm{km}^2$	$4.5\,\mathrm{km}$	$11.8\,\mathrm{km}^2$	$1.9\,\mathrm{km}$

Table 4.4: Estimated surface areas of Voronoi polygons, which approximates the coverage of the cell towers

journalist-whistleblower communication. In subsequent chapters we look at the higher privacy needs journalists require to communicate securely with their sources (which may later become whistleblowers).



(a) Direct friends



(b) Friends of friends

Figure 4.8: Histograms of the co-locations with (a) direct friends and (b) friends of friends as calculated for every 10-minute interval in every cell tower location.

CHAPTER 5

UNDERSTANDING WHISTLEBLOWING AND THE ASSOCIATED RISK

The Snowden leaks [90] revealed just how widespread mass surveillance is. It is not only a significant breach of privacy for the general population, but it also poses significant danger to select groups, such as whistleblowers. In the recent past, whistleblowers have not only been fired [172] or suffered professional boycotts [196], but also suffered more severe punishments, such as torture, imprisonment [208, 230] or were even killed [207]. However, whistleblowing is pivotal to holding those in power responsible when illegality, wrongdoing or abuse of power happens [14, 90, 176], and must be adequately supported by deployed systems and the research community. Unfortunately, computer security research into systems suitable for whistleblowing is sparse [142]. We therefore take a complementary approach since whistleblowing happens regularly: we review systems currently in use by journalists (Section 5.1), review the security offered by those systems (Section 5.5), and organise two workshops with major British news organisations to understand the space (Sections 5.2 and 5.3). The first workshop aims to understand the needs of our primary users – the investigative journalists – and the second workshop presents and sanity-checks the technical requirements we derived from the ideas drawn from the first workshop to a more technical audience from one of the news organisations that attended our earlier workshop.

We find that the risk of identification for a source is great when an inadequate technology is used to communicate with a journalist. Unfortunately, current available technology addresses specific issues and has different goals and threat models depending on the target user. These methods are not directly applicable to the needs of journalists and their sources. Furthermore, there is a strong imbalance between security and usability in those systems, as often one fails in favour of the other. The lack of systems deployed for this user group leaves journalists and their sources exposed to significant risk.

This chapter, therefore, focuses on understanding the communication needs of journalists and their sources, presents the motivation for our focus on whistleblowing systems, and, particularly, our decision to secure the initial contact between journalists and their sources. Based on this research we build a system called CoverDrop, which we present in Chapter 6.

This chapter is largely based on peer-reviewed research published in the Proceedings on Privacy Enhancing Technologies Symposium 2022 [7]. The initial idea for providing a secure method for users to communicate with journalists spawned from several discussions between myself and Alastair on the subject of inadequate communication security protections in journalism. We originally scoped out the idea of including a secure communication app, such as Signal, within the news app, which was the prime motivator for the work. Upon Mansoor joining the team, we refined details of the messaging app (which we later named CoverDrop) as a group, such as using Trusted Execution Environments to provide metadata protections and Secure Element on devices. We presented this CoverDrop system overview for our audience at the workshops presented in this chapter as a driver to generate ideas and allow us to ask relevant questions. Details of the final CoverDrop system which we present in the next chapter we finalised upon the analysis of the workshop discussions. My contribution relevant for this chapter is, therefore, varied throughout the project ranging from ideas generation, to running the workshops, and transcribing and coding the results. I initially investigated the contact options provided to journalistic sources by analysing the websites and apps of 24 different major newspapers across the globe, as well as studied the main communication options that have been used in the past. I also led the organisation of two workshops with major British news organisations and transcribed the recordings of the first workshop together with Mansoor. Furthermore, I was primarily responsible for analysing the transcripts and drawing the major points from the workshop discussions, which were later corroborated by Mansoor's analysis. This research both shaped our understanding of the user requirements for a whistleblowing application, and led the formulation of the technical and security requirements of the CoverDrop system, which are later presented in Chapter 6.

In summary, this chapter provides the following contributions:

1. an analysis of current options available to secure the journalistic communication process;
2. an understanding of journalist processes and technical requirements for a system to secure communication between journalists and their sources, through two workshops with British news organisations;
3. an understanding of the adversarial model faced by whistleblowers and journalists; and
4. an analysis of related work and the security of systems currently in use by journalists.

5.1 Contact options for major news organisations

We start by reviewing the recommendations made by 24 major news organisations to possible sources when they wish to contact a journalist with a story. We chose 24 news organisations from around the world based on their high scores on Alexa top sites, ranked by category and country. We begin from the web home page of each news organisation and investigate the options suggested to people to contact them by following the "Contact us" pages. The results of our review and our analysis of the security properties offered by these options can be found in Table 5.1, ranked by popularity which is measured in terms of news app installs on Google Play. We chose the popularity metric based on the number of app installs on Google Play rather than the Alexa ranking as it is relevant to the approach chosen in designing CoverDrop as a news app library.

Our findings show that only half of the newspapers offer an encrypted means of communications. This is either a general-purpose encrypted messaging app, or a special-purpose system such as SecureDrop. Furthermore, only two newspapers offer in-house, tailored, secure technology, which is unfortunately closed source. Commendably, two of the reviewed newspapers, The New York Times and The Guardian, offer easy-to-understand information on the suitability of the different options provided based on the needs of the user.

We believe such important information should be advertised often and should be easy to find on a newspaper's web page, such that a potential whistleblower does not need to search for key words, thus exposing themselves to further risk. Therefore, in our analysis, we derived an "easy to find" metric by counting the number of links a user has to traverse from the homepage in order to reach the relevant information needed to initiate contact, particularly based on the fact that a potential whistleblower should not just search the web for such information. We find that most newspapers score two or more. Some of the ones that score one, had direct links to special-purpose systems such as SecureDrop, which may not be directly obvious to the potential source. Notably, The Guardian, not only had a link one hop away, but also regularly run a banner through the middle of the page as the user scrolls news articles advertising methods to initiate contact.

Another area of interest would be whether sources typically try to contact a member of staff directly or whether they would go through a main triage entry point. We find through our review that only a few newspapers offered PGP keys directly for staff, and most just offer company-wide or department-wide contact points. This was validated by the journalists in our workshop (Section 5.2) and was incorporated in our technical requirements for CoverDrop (see Section 6.1).

Newspaper	Popularity	Easy to find	Postal mail	Telephone	Plaintext email	Online form	Messaging app	Encrypted email	Tor/SecureDrop	Other	Relevant page
New York Times	10m+	2*	●	×	×	×	●	●	●	×	[209]
CNN	10m+	4	×	×	×	×	×	×	×	×	[36]
Times of India	10m+	×	×	×	×	×	×	×	×	×	[162]
BBC	10m+	×	×	●	●	●	●	×	×	●	[152]
The Guardian	5m+	1*	●	●	×	×	●	×	●	●	[91]
Spiegel	5m+	5	●	●	×	●	×	●	●	×	[164]
Le Monde	5m+	5	×	×	×	×	×	×	×	●*	[149]
Washington Post	1m+	×	●	×	●	×	×	×	×	×	[171]
El Pais	1m+	×	●	●	×	×	×	×	×	×	[166]
Süddeutsche Zeitung	1m+	2	●	×	●	×	●	●	●	●	[204]
Wall Street Journal	1m+	×	●	×	●	×	×	×	×	×	[110]
The Mainichi Sh.	1m+	×	×	×	×	●	×	×	×	×	[135]
The Sun	500k+	5	●	●	●	×	●	×	●	×	[202]
China Daily	500k+	×	●	●	●	×	×	×	×	×	[47]
O Globo	100k+	×	×	×	×	●	×	×	×	×	[80]
Buzzfeed	100k+	2	●	●	×	×	●	●	●	×	[154]
Globe and Mail	100k+	1	●	●	●	×	×	●	●	×	[79]
Dawn	100k+	×	×	●	●	×	×	×	×	×	[50]
The Sydney M.H.	50k+	×	×	×	●	×	×	×	×	×	[94]
Wired	10k+	1	×	×	×	×	×	×	●	×	[231]
The intercept	×	1	●	×	×	×	●	●	●	×	[102]
Wikileaks	×	2	×	×	×	×	×	×	●	×	[229]
Private Eye	×	×	●	●	●	×	×	×	×	×	[68]
Ars Technica	×	2	●	×	×	●	●	●	×	●	[16]

Table 5.1: A view of 24 different newspapers across the globe and the options they provide sources to contact the editorial team. Popularity is calculated by the number of installs from the Google PlayStore. Easy-to-find is calculated as the number of hops from the main web page. *Legend: ● = offered*; *× = not offered*; ** = custom solutions offered*

5.2 Workshop 1: Understanding the journalistic process

To better understand the journalistic process, in particular what the journalists' priorities are and how current procedures and system options clash with these, we organised a workshop with journalists and information security experts from several major British news organisations.

5.2.1 Ethics considerations

We obtained approval to run the workshop and to collect the results from the Department of Technology and Computer Science at the University of Cambridge Ethics Committee. We also sought approval from our participants to record the discussion during the workshop in order to produce a transcript that can be analysed. We kept participant identity and affiliation anonymous [99] by coding each person as P1 through to P20 when transcribing the discussions from the workshop.

5.2.2 Workshop format

The 4-hour workshop was held in London in September 2019. In total we had 20 attendees (coded as P1 through to P20): 17 in-person, 3 call-ins. During the workshop we presented the participants with our view of the current world (Table 5.1) and an initial idea for the CoverDrop system. This was aimed at generating discussion. We also used a semi-structured approach, commonly used in interviewing, by having a list of open-ended questions to drive discussion and challenge our understanding. These questions covered areas such as: source cultivation processes, current systems being used and what are downsides of those systems, keeping sources and journalists safe, keeping data safe, and enabling secure collaboration between journalists.

5.2.3 Transcript analysis

The primary aim of this workshop was to gain an understanding of current systems and processes in use by journalists to communicate with sources, as well as define the user requirements for a suitable replacement. The most important result of the workshop was that there is not a system that can fit all cases, purely due to the continuously changing nature of requirements. Furthermore, some of the mainstream applications that offer good usability are not great at providing anonymity or even metadata security guarantees, whereas those intended to provide such security goals are subject to usability issues, such as high latency or expert knowledge requirements. However, it helped us identify a soft-spot: *a system to secure initial communication with a journalist*, which can then be used to complement secure data transfer systems such as SecureDrop or PrivNote.

To analyse the results, we conducted a thematic analysis over the transcript using double-blind coding. This helped us observe recurring themes in the discussion and define our understanding of the needs of journalists and their sources. We used Workshop 2 (Section 5.3) to validate the themes that emerged from Workshop 1.

The journalistic processes

Due to the restricted amount of research on journalistic processes that are relevant to the computer security community, we dedicated a significant amount of time to validating and complementing our understanding of the process that follows from an initial contact to a story being reported. Below are main themes that emerged from the discussion relevant to the journalistic process:

- **Source cultivation is a long process.** Building trust takes time. And this trust is central to a journalist-source relationship because *"[sources] want to build up the trust relationship before [sharing the sensitive information]"* (P4).
- **Communication importance evolves.** Often contacts start a conversation in an insecure channel because *"it is rare for it to be a high degree of emphasis placed on security by the source at that stage"* (P1). As the communication evolves, so does it's importance and then the conversation *"moves to another channel"* (P4). So, as the source is getting increasingly worried about their job security, *"they tend to become conscious about security"* (P1). However, at this time a change in platforms attracts attention and poses a direct threat to the source's anonymity.
- **Sources often change in nature.** Not all sources start out with the intention to blow the whistle: *"Confidential sources sometimes become whistleblowers because they're getting increasingly angry about something"* (P8).
- **Reputation is important.** *"[Being contacted by new sources with information is] not just direct"* (P4). A journalist-source relationship isn't always defined as either having some form of prior contact, or being complete strangers. Sometimes sources come through recommendations and a journalist's reputation plays an important role: *"the hope is that we can proactively approach an individual with the story based on our reputation for covering that area. Or, alternatively, our reputation is such that people might wish to contact us directly"* (P9).
- **Journalists take responsibility.** Dealing with communication security falls on the journalists who *"take responsibility [as to what secure tools to use] upon themselves a lot of the time"* (P9). Furthermore, because popular insecure technology has made it easier to identify sources, it is now easy to *"identify 5 people who at some point have exchanged emails [with the journalist or the newspaper] and that's the sort of retrospective footprint problem"* (P8). Dealing with the aftermath of insecure platforms is another hard problem often left for the journalist to try and mitigate because *"the damage is done in security terms"* (P1).

The systems

A significant part of our workshop was dedicated to discussing current options, including the benefits and drawbacks of the systems currently available to journalists, as well as understanding what risks are considered acceptable. From the beginning our attendees remarked on the importance of whistleblowing, despite it being rare, with *"less than 10%"*(P4) of user-sent content ever becoming a story, yet *"you cannot give up the opportunity that some day someone will"* (P13). The following themes thus emerged about current options, which we used to guide our choices for CoverDrop (see Chapter 6):

- **High latency.** The journalists remarked that current solutions are slow. For example, SecureDrop *"is very high latency. [...] someone has to download [messages] on a USB stick and take it to an air-gapped environment and then decrypt it. And even that is quite a slow process."* (P4). Through discussions we learned that the manual workload involved in the decryption of SecureDrop messages on an air-gapped system increases the system's latency to a magnitude of around a day. Having a lower-latency system to enable multiple messages (back and forth) is important in the early stages of trust establishment.
- **No message receipt confirmation.** The fact that most secure (purposely-built) systems do not handle message receipt confirmations, coupled with the real-world high latency, may cause problems to an already-worried source. We were told some work-arounds exist: *"So we actually somehow mitigate the latency: we send an automatic response when we receive stuff, to let people know it has been received on the server, even though we haven't actually read it, or downloaded it, or decrypted it. Just to reassure people."* (P4)
- **Difficult to use or understand.** Lastly, and one of the most important observations was that a good balance between usability and security is important: *"I think we'd like the easiest solutions that give us with usability the best protections"* (P13). This remark is strengthened by the fact that the attendees showed a clear grasp of both confidentiality and anonymity: *"[we] distinguish quite a bit between confidentiality and anonymity, just cause the latter is so much harder"* (P4).

Furthermore, we learned that it is worthwhile having a front-desk approach for incoming messages, as well as individual contact points for journalists themselves: *"a lot of softer stories tend to come [through a front desk]. Bigger stories, in my opinion, do not tend to happen through the front end, but that might be with how we're set up"* (P1). Lastly, resilience to warrants is important for such a system. Many participants commented on the fact that a trusted whistleblowing system would enhance transparency and lead to a better democracy.

5.3 Workshop 2: Technical requirements

We were invited by one of the organisations attending Workshop 1 to deliver a presentation of CoverDrop to several departments of their news organisation. We used this opportunity to validate the themes that emerged from the analysis of the Workshop 1 transcript (Section 5.2), as well as to derive and confirm the main technical requirements of the system we were building.

5.3.1 Ethics considerations

To attend this workshop, present our research and take notes of the discussion, we obtained prior permission from the Department of Technology and Computer Science at the University of Cambridge Ethics Committee.

5.3.2 Workshop format

This workshop differs both in goals and format to its predecessor. This workshop was more informal, approximately 2 hours in length and the list of attendees was refined by the news organisation. We presented our research to a team of approximately 40 people from teams in software engineering, system administration, and, of course, investigative journalism. The presentation was followed by free discussion on technical aspects of CoverDrop.

5.3.3 Analysis

The analysis following this workshop is based on the researchers' personal notes from the meeting, which were then corroborated amongst the team members. From Workshop 1 we defined the requirement: a secure, anonymous and low-latency initial contact method for sources. This is designed to help bypass the initially insecure options currently available, as well as remove the need for dealing with the "retrospective footprint". We used this workshop to validate the themes that emerged through Workshop 1 (Section 5.2). As such, it was confirmed that a high priority for any system is the ease of deployment. Many participants confirmed that a higher success rate is guaranteed when accompanied by minimal engineering effort. Furthermore, we received interesting insights into important technical aspects, such as a news website's (or app) dependence on Content Delivery Network providers (CDNs) and how to navigate the complication added by CDNs to our proposed solution, the risk posed by external libraries that are part of the news app, as well as the risk posed by ads. We used these insights to drive the development of our CoverDrop system, as well as the prototype. Lastly, several attendees stressed the importance for news organisations' app developers to be able to patch the app when vulnerabilities are found.

5.4 Adversarial assumptions

We used our research of existing systems together with the results from our two workshops to arrive to a realistic adversary model. To build our adversary model, we begin by summarising the main failure points in the communicating devices of a journalist and a source.

5.4.1 Possible points of failure

The most common points of failure for the communication between a whistleblower and a journalist lie in one of four options: device compromise, network compromise, infrastructure compromise, or end-user security mistakes.

- **Device compromise.** Mobile devices (such as tablets or smartphones) can be compromised in multiple ways. *Spyware apps without root privileges*, for instance, can record the screen, however this cannot be done discreetly due to OS warning notifications being triggered when this happens. Side-channels that allow the partial capture of keystrokes [33] can be safely excluded from the threat model since their signal-to-noise ratio is too poor to enable police surveillance. *Root spyware*, on the other hand, enables a wide-range of access such as data stored by other apps, UI control and keystroke logging, which would render network security protections ineffective. We consider root spyware out of scope for our threat model. Recent research showed that root spyware is not completely undetectable [103], and should this be a concern, our recommendation is to only use CoverDrop for the initial form of contact, upon which getting a burner phone from the journalist and overwriting the active CoverDrop session. Lastly, we assume *temporary adversarial possession* is possible and include it in our threat model. In this case, USB forensic devices may be used to obtain system information, provided the phone had been unlocked since it was powered on. As such, we assume that the Secure Element (SE) or equivalent Trusted Execution Environments (TEEs) can resist such methods.

- **Network compromise.** Whistleblowers can face incrimination if it can be proved that communication took place between them and a journalist at the relevant time. As such, there is a need for completely unobservable communication: protecting both message contents and any associated metadata (thus hiding that communication even took place). We assume our adversary can access any relevant networking infrastructure such as CDNs, ISPs, or even corporate LAN. However, we assume the adversary is a passive observer aiming to infer connections between users, rather than attempt to drop or manipulate messages.

- **Infrastructure compromise.** Warrants are commonly used by law enforcement to gain access to systems and data relevant to an ongoing investigation. Therefore, we assume that warrants may be issued after the individual blew the whistle. These warrants could be served to any third party, including ISPs, post office, or cloud providers, as well as the news organisations themselves.

- **End-user security mistake.** Users are central to the correct operation of security systems. Even the most secure systems fail if usability was not at the forefront of the system's design [182, 192, 227]. When people are under stress, or they are doing something new, or it is something they do rarely, the risk of operational mistakes increases [6].

5.4.2 The adversary

We arrive to the following baseline adversary model which we believe is realistic for a government whistleblower wishing to remain anonymous. In Section 6.3 we discuss stronger adversarial models. Therefore, our baseline adversary can:

1. monitor and record any communication to and from a news organisation, the journalists and any potential sources;
2. monitor and record any network traffic on the news organisation internal network;
3. issue warrants either during or after the leak took place, therefore gaining physical control of devices and servers;
4. issue warrants to any third parties (CDNs, ISPs, cloud services, or messaging app servers);
5. install non-root spyware apps on journalists and possible sources' devices; however,
6. they cannot compromise the confidentiality and remote attestation guarantees of a TEE during normal runtime operation. Should a warrant be issued or physical attacks performed post-disclosure, we cannot assume any TEE guarantees. Lastly,
7. they cannot compromise the smartphone's SE, even when in its in their physical possession.

5.5 Related work

Research in the area of secure communication for journalists and sources is sparse. In this section, we split the related work analysis into two separate areas: an analysis of the research space, and an analysis of the systems currently used by journalists and sources to communicate (whether secure or not).

5.5.1 The research space

The research space is multi-disciplinary. On the one hand, journalists hold insufficient security knowledge to reasonably provide safety for their sources [206, 212]. On the other hand, the security research community needs a better understanding of the needs of investigative journalists, their workflows and processes, in order to address the significant barrier to secure and usable communication that journalists and sources, alike, are facing [142]. While this is a long-standing problem, the Snowden revelations, which revealed proof of mass surveillance and abuse of power [90, 93, 124], brought more attention to communication security [126], particularly to avoid the unwanted effect of silencing any future whistleblowers [224]. Unfortunately, this has complicated investigative journalism because there is a lack of consensus on what is needed to establish secure (and possibly prolonged) communication between a journalist and a source [220]. Protecting confidential sources is important to maintain freedom of the press, as well as to expose any wrongdoing inside governments and organisations [148], however this often results in conflicting and independent choices made by the journalists themselves to maintain source security [220].

Ad networks can be used to route messages from a source to a news organisation [181], however, this type of submission platform cannot guarantee message delivery, nor does it withstand our adversary. Keybase has also been used to provide the key verification mechanism for an encrypted email platform, Confidante, which is designed to support journalists and lawyers [130]. However, due to compatibility issues with email, Confidante does not attempt to hide metadata, which does not support our adversarial model. Using Tor to upload documents to news organisations [106] has both advantages and disadvantages when it comes to security and anonymity. Particularly, because it requires computer security knowledge for a user to operate securely, as well as the fact that Tor is banned in some countries. Furthermore, simply using Tor may act as a red flag, e.g. if in a company of 4 000 employees only a handful use Tor, it greatly reduces the list of possible suspects. With this in mind, we designed CoverDrop to have an easy-to-use secure initial contact method for sources to communicate with journalists (see Chapter 6).

5.5.2 Systems in use

While the research space is sparse, journalists do communicate often with sources in practice (some of whom may become whistleblowers). We, therefore, review the systems largely in use today and reason on the level of security offered to users. We focus on the main contact method extracted from our research (Table 5.1), which were confirmed by the workshops we organised (Section 5.2 and Section 5.3): physical mail and unencrypted email, secure messaging, encrypted email, and SecureDrop.

Physical mail and unencrypted email. These are perhaps the easiest to understand and use, which is why they are the most common method of contact (Table 5.1). The source, without any specific security training, can take any precautions they deem fit when using physical mail. For instance, not adding a return address attempts to secure the source's location, but it makes communication one-way, and through postage franking one can still estimate geographical locations. Furthermore, widespread video surveillance makes any letter or package drop-off risky, and delivery is not guaranteed. Similarly, with unencrypted email, the ubiquity of free email providers makes it easy for a source to create and use a single-use email address for this purpose. This solves some of the issues with physical mail, making the communication two-way, and fast delivery. However, it exposes the user because of the metadata logging email providers perform, and it has a lack of in-built encryption by default.

Encrypted email. Encrypted email is one way to combine the benefits of encryption and the ease of message transfer offered by the email servers. Unfortunately, encrypting email content through the use of **PGP** is known for it's notorious usability issues [76, 182, 192]. Even without the usability problems, metadata logging would still be problematic, thus enabling correlation attacks, as well as the fact that most modern email providers now require a phone number or recovery email address when creating a new email address with them.

Secure messaging. End-to-end encrypted apps such as **Signal** and **WhatsApp** have risen in popularity in the past years and are now actively and securely used by billions of regular users daily. While Signal is partially open-source, WhatsApp is closed-source, but shares code components since it used the Signal messaging protocol as a basis to deploy end-to-end encryption. Despite the ease-of-use and security provided, which were missing from the previous options, both apps have a common weakness — mobile phone numbers are used as the main identifier for user accounts. So, while guaranteed to be fast, encrypted message delivery in a two-way channel, a source has to reveal their phone number to the journalist in order to initiate contact. Furthermore, there is no network obfuscation, so a passive global adversary could perform a timing analysis. Since ISPs already log message metadata, such as sender and receiver IP addresses, a less powerful adversary could use this information as an alternative attack vector. Lastly, unlike Signal, WhatsApp message metadata is collected and centrally stored by default, so information such as time of communication and the message size would be subject to subpoena, without either party being aware of this (because typically a gag order accompanies a subpoena prohibiting the service operator from notifying the users).

SecureDrop. As a service purposely built to protect whistleblowers' identity by allowing them to share files with reporters anonymously, SecureDrop targets a comparable user space, security goals, and threat model as our system, CoverDrop (the name itself is an homage to SecureDrop). It achieves anonymity by generating pseudorandom identifiers for the whistleblower and the account created is not linked to any personally identifiable data such as a phone number, name or banking details. It runs over the Tor network by relaying all the messages received so that it obfuscates the whistleblower's IP address. However, simply downloading and using Tor can single out a user and has previously put people on NSA watchlists [180], so the fact that a whistleblower might be using Tor for the first time as an artefact of using SecureDrop would reduce their anonymity set gravely (if the whistleblower is the only Tor user in the office from where the leak originated, then it may cause trouble). Furthermore, securely installing and browsing the Tor network (i.e. to navigate to the newspaper's onion address) requires a high level of skill, thus posing usability problems for the lay person. Lastly, our workshop attendees who used SecureDrop revealed that the high latency of SecureDrop processes is a real problem for a possibly nervous source. Our workshop discussions revealed that the real-world round-trip latency of SecureDrop is in the order of days, and coupled with a lack of read receipts has led to anxious sources dropping out. A recent trial of SecureDrop Workstation aims to improve this latency by using Qubes OS to open and decrypt each message into an offline virtualised environment allowing it to obtain compartmentalisation, thus achieving comparable security while removing the latency involved in using an air-gapped workstation [189].

Other options. Our review of current options presented to sources showed other options are used, albeit rarely (Table 5.1). For instance, another secure messaging app, Threema, is different from the other options presented above because it does not require a phone number or email address to sign-up. However, the app requires payment through the app store, thus raising other privacy concerns. Another type of anonymous data transfer similar to SecureDrop is PrivNote. It enables anonymous and encrypted data transfer to a dead-drop site, but it's main differentiator is that through its simplicity it only provides one-way communication. This was a crucial system capability to enable trust establishment in the source — journalist relationship. Furthermore, it requires a secondary means of communication to enable the source to share the link to the dead-drop site with a journalist — an operation where CoverDrop might be ideal.

Lastly, in high-profile cases, where the threat is extreme, root malware becomes an option as it can be installed remotely and covertly on the devices of all suspects and reporters. While this is rare due to the rarity of zero-day exploits, they are difficult to block. Burner phones are used in high-profile cases to enable the source and the journalist to communicate on devices unknown to authority. The role of CoverDrop is to simplify and secure the initial contact establishment, thus helping the news organisation to become the destination of choice for future Snowdens.

5.6 Summary

This chapter discusses the risk involved in whistleblowing, analyses what is currently used in practice, as well as any related research exploring the secure journalism space. We organised two workshops in London in 2019 with attendees from major British news organisations and summarise the results, which helps us draw out important aspects of journalistic workflows. We use the information learned from the workshops to describe a realistic adversarial model for a government official who wants to anonymously blow the whistle. The work in this chapter helps us understand the computer security research opportunity in journalist-source communication, which is a method for a source to securely and easily make contact with a journalist. In the next chapter we introduce CoverDrop, a system that was developed to address the issues faced by journalists trying to retrospectively protect their sources by enabling news readers to establish this initial contact with the news organisation (particular journalist or the front office) in a secure and anonymous way through the news app already installed on their phone.

CHAPTER 6

CoverDrop: securing a source's first contact through a news app

In Chapter 5 we discussed the problems journalists commonly encounter when trying to protect their sources and reviewed currently available solutions. Through our workshops we identified that one communication problem journalists have is insecure initial contact, particularly because it is a much harder problem to apply the necessary protections retrospectively to fix any footprints left by insecure means of communication. In this chapter, we present our solution to this problem — CoverDrop — a simple and secure two-way communication platform built within a news reader app. CoverDrop protects the communication between a source and a journalist against the powerful adversary we identified in Section 5.4.2. CoverDrop can be integrated with Content Delivery Networks (CDNs) and uses cover traffic generated by all the users of the news app to mask any communication taking place between a source and a journalist (or the news organisation) and thus, benefits from the large user-base news reader apps typically have. The traffic (whether real or not) passes through at least one Threshold Mix (called a *CoverNode*) upon reaching the news organisation's network. Another advantage of using CoverDrop directly in a news reader app over other systems is that the source does not need to install another app on their device, which acts as a give-away sign of their intent or actions.

The work in this chapter is largely based on peer-reviewed research published in the Proceedings of Privacy Enhancing Technologies Symposium 2022 [7]. The initial idea came about through discussions between Alastair and myself. Since then, CoverDrop has been the product of team effort and the final design is the result of all co-authors: Mansoor, myself, Daniel, Ross and Alastair. My contributions include work on the overall design, with a particular focus on building a system consistent with the workshop feedback (see Chapter 5). I also analysed overall the security goals and system limitations, and performed most of the security analysis with contributions from Daniel reflected in analysing the unobservable communication goal and the plausible deniability goal when the device is confiscated. Mansoor implemented the SGX-based

CoverNode; and Daniel implemented the sample news app and the CoverDrop mobile library, as well as ran the performance analysis and performed a semi-formal security analysis of the properties offered by the encrypted storage on the mobile device. The figures used in this chapter were designed by Daniel (the protocol flow in Figure 6.1 and the UI screenshot in Figure 6.2) and I refer to them to clarify system details. Similarly, Table 6.1 was designed by Mansoor and I refer to it to compare CoverDrop's features to other systems currently used for whistleblowing.

In summary, the contributions made in this chapter are:

1. A secure mobile library for easy deployment of covert communication channels masked by cover traffic in existing applications.

2. A TEE-based CoverNode that provides strong protection against a powerful adversary and supports 3 million messages per second per CPU core.

3. A networking model that supports integration into existing CDN-based networks with minimal modification.

4. Strong protection against a global passive adversary that can also use warrants to compel infrastructure providers and to seize devices from journalists (and their sources).

5. An open-source prototype implementation.

6.1 CoverDrop overview

CoverDrop is a messaging system designed to help the users of a news reader app initiate secure contact with news organisations (either through a front desk approach, or direct contact with a journalist). We review CoverDrop's system requirements as well as design decisions, which were guided by our workshop with British news organisations (see Chapter 5).

6.1.1 System requirements

During our workshops we identified problem areas in existing systems as experienced by the journalists and information security experts attendees. Identifying these problem areas helped us develop a set of functional requirements for CoverDrop:

- **Two way asynchronous communication.** It was apparent in the workshop that two way asynchronous communication resembling popular messaging apps is essential to support the initial stage of communication in which the correspondents build trust with each other through several message exchanges and answer each other's questions.

- **Low latency.** An important feature is low latency to allow multiple messages to be exchanged daily. This supports the early stages of trust establishment between a source and a journalist since whistleblowing tends to require multiple rounds of communication and high latency can cause additional stress to an already stressed source.

- **Responsiveness.** A high level of responsiveness from the chosen method of communication, such as message delivery confirmation, is preferred to provide peace of mind to anxious sources.
- **High usability.** The UI should be easy to use for all users and should not expect any specific cryptography knowledge in order to offer a high level of security.
- **Ease of integration.** A solution that is easy to integrate, both for the news organisation's network and the news reader app.

We note that in CoverDrop we favour low latency over high bandwidth, which makes CoverDrop a suitable system to initiate contact, establish trust, and to decide on next steps when the source is ready to share files. Thus, CoverDrop can be seen as complementary to systems such as PrivNote and SecureDrop.

6.1.2 Design decisions

The two workshops we organised with British news organisations helped us to better understand the requirements and priorities in this space (see Chapter 5). Thus, we arrived to our motivation, centred around two driving ideas: (1) existing systems are inadequate in the face of a strong adversary (defined in Section 5.4.2), and (2) usability is key to security. As a follow-up to the workshops it was evident that several design decisions need to be addressed for: the infrastructure, the security offerings, and the user interface. In this section we discuss the main design decisions we took for CoverDrop and the benefits or disadvantages of each.

When it comes to *the infrastructure*, we tackled two main requirements in CoverDrop's design: not requiring any additional software to be installed by the end-users, and reducing the integration efforts for the news organisations. To not require any additional software for end-users we chose to design the CoverDrop functionality directly inside existing newsreader apps. The advantage of this approach is two-fold: we benefit from a large existing user base of the newsreader app (see Table 5.1, which details the number of Android installs of each app), and we remove the initial giveaway of intent for a user who installs custom whistleblowing software (e.g. if the leak originated from a particular office, a passive adversary can observe which employees downloaded a Tor client and used SecureDrop). We introduce constant-throughput channels (see Section 2.5.3) in our design to benefit from large numbers of active users such that each regular user's app (those that installed the news app) sends a constant amount of cover traffic, which allows us to create cover traffic to hide when real conversation takes place. Such messages have two layers of encryption applied to them (inner encryption for the recipient and outer encryption for the proxy handling the bulk of incoming messages), such that an outsider cannot distinguish between real or cover messages. The immediate downside is throughput is limited to short text-only messages (see Section 6.1.1 for justification) making CoverDrop suitable for text messaging rather than large file transfers.

Furthermore, the newsreader app interacts with the news organisation's infrastructure, typically relying on a Content-Delivery Network (CDN). To enable ease of integration in the news organisation's app infrastructure, we designed CoverDrop to easily integrate with the organisation's existing CDN infrastructure, which can route both real and cover messages.

Our *security-related choices* for the type of conversation that takes are based on the following flow: a source (whistleblower) decides to contact a news organisation or a particular journalist with a lead for a story; therefore the source initiates the contact with an initial message. This flow allows us to have a system in which journalists and theme-based newsroom desks publish public-keys to enable the encryption to automatically take place for the user's message. Based on the chosen recipient, the app forms the inner encryption layer, to which it applies an outer encryption layer for a constant-throughput proxy, whose design we detail below.

We use constant-throughput proxies and double encryption to thwart correlation attacks and ensure that real messages cannot be distinguished from the constant throughput of messages (cover or real). Our constant throughput proxy is called a CoverNode and it is implemented within a Trusted Execution Environment (TEE). All the real messages have two layers of encryption: the inside layer encrypted with the recipient's public key and the second, outer layer encrypted using the CoverNode's public key. The CoverNode is then used to decrypt the outer layer of encryption within the TEE to distinguish real from cover messages and drop the IP address of those who send real messages to ensure such information is not accessible. In our system, we include one CoverNode, but for added protection we highlight the changes needed to enable multiple CoverNodes operating across multiple news organisations (see Section 6.2.4).

For added protection against device seizure or spyware, we store all the app data, which includes any active chat logs, encrypted and padded to a fixed size, regardless of whether the user has used the CoverDrop functionality of the news reader app or not. This ensures plausible deniability for all users who have the app installed. Furthermore, we derive the master secret used for storage encryption from the passphrase provided by the user and a secret stored in the Secure Element (*SE*) of the device.

Lastly, when it comes to *the user interface*, we designed CoverDrop with the look and feel of modern messaging app to enable even the less experienced users to operate it successfully. We provide additional assurance for users (when messages are due to leave the device, or when the next incoming batch of messages is to be expected) within the chat log.

6.1.3 Security goals

Our workshops (Section 5.2 and Section 5.3) revealed three core security properties required for making initial contact with journalists: confidentiality & integrity, unobservable communication, and infrastructure plausible deniability. We outline CoverDrop's security goals with reference to our adversary model (Section 5.4.2) and justify in Section 6.3 how CoverDrop achieves these goals.

G1: confidentiality & integrity. The adversary is unable to read or modify the contents of messages between potential sources and journalists on any computer network.

G2: unobservable communication. The adversary cannot tell which, if any, news app users are currently sending messages to journalists.

G3: plausible deniability. Even with physical possession, the adversary cannot determine which, if any, news app users have previously used CoverDrop.

6.1.4 Limitations

There are inherent trade-offs between the provision of cover traffic, fixed-size messages, and rate limits. An increase in bandwidth would allow us to support larger or more frequent messages for the minority of users who are whistleblowers, but require the remaining users to send larger volumes of cover traffic. However, our research has determined that there is a sweet spot where the traffic costs for the average user remain low while the bandwidth required to support whistleblowers is sufficient to support first contact with a journalist. We note that messages larger than the predefined size are split into multiple packets and sent over time (due to the rate limits).

We leave for future work the inclusion of stronger privacy guarantees such as forward secrecy and post-compromise security, which are becoming the norm for secure messaging apps, to retain protocol specification simplicity. During implementation, one could achieve forward secrecy, for example, through small changes to the protocol, such as adding session identifiers and key state, or rotating public keys.

Lastly, to fully support security goals G2 and G3, CoverDrop relies on the security of TEEs (in our implementation SGX), Secure Element on device, and third-party libraries. We discuss the effect compromise of either of these elements has on the security of CoverDrop in Section 6.3.2 and provide an extended analysis of the properties provided by encrypted storage on the mobile device in our publication at PETS'22 [7].

6.2 System details

In this section we provide the system details for CoverDrop by first detailing the actors involved in the protocol and discussing the protocol flow before looking at any individual components.

We exclude implementation details for which we have provided the open sourced code[1].

6.2.1 Actors

CoverDrop takes four different types of actors into account, whom we detail in this section.

News organisations have a vested interested to provide secure and accessible ways for their readers to contact their journalists with leads for stories.

Journalists have a vested interest in making a source comfortable sharing news-worthy leads or documents. Part of making sources feel comfortable is also an assurance of protections, thus journalists often take matters into their own hands to protect their sources (see Section 5.2).

Sources may indeed turn out to be *whistleblowers*. They want to initially make contact in order to share some information with a journalist, however they would like to maintain their anonymity.

Regular users of the news reader app expect to continue enjoying their app and not be noticeably affected by the pressures of increased energy consumption (affecting their battery performance) or increased mobile data traffic (affecting the price they pay for their mobile data plan).

6.2.2 Protocol flow

For the protocol walk-through we refer to Figure 6.1. We discuss the components and processes involved in the CoverDrop protocol flow, which includes the deployment of CoverDrop, the enrolment steps required for journalists and users, what the app does in the background to generate cover traffic, and what happens when a CoverDrop session is started (i.e. the process of sending a message) and messages are being passed between journalists and sources.

There are three sets of public-private key pairs we will use in this section: journalist app (pub_J and $priv_J$), the CoverNode (pub_{SGX} and $priv_{SGX}$), and the source app (pub_A and $priv_A$).

We differentiate below between the protocol flows for the journalist app and the source app. The first step for any journalist app is **enrolment**. When CoverDrop is initially set up all the journalists choose a unique, fixed-length identifier (id_J), as well as generate a public-private key pair (pub_J and $priv_J$). To enrol in CoverDrop, the journalists send to the CoverNode's SGX enclave a signed tuple $\{id_J, pub_J\}$ ⓐ. Should the news organisation already be using an Identity Provider for their accounts, the enrolment process can be modified to use it instead. The CoverNode's SGX enclave uses the signed tuple by first verifying the signature and then adding the tuple to the map of journalists $journalists : \{id_J \rightarrow pub_J\}$. These mappings are periodically forwarded by the SGX enclave together with the SGX public key to the WebApi/CDN: `POST /pubkeys` ⓑ.

[1]Github source code:
`https://github.com/coverdrop/prototype/tree/main`

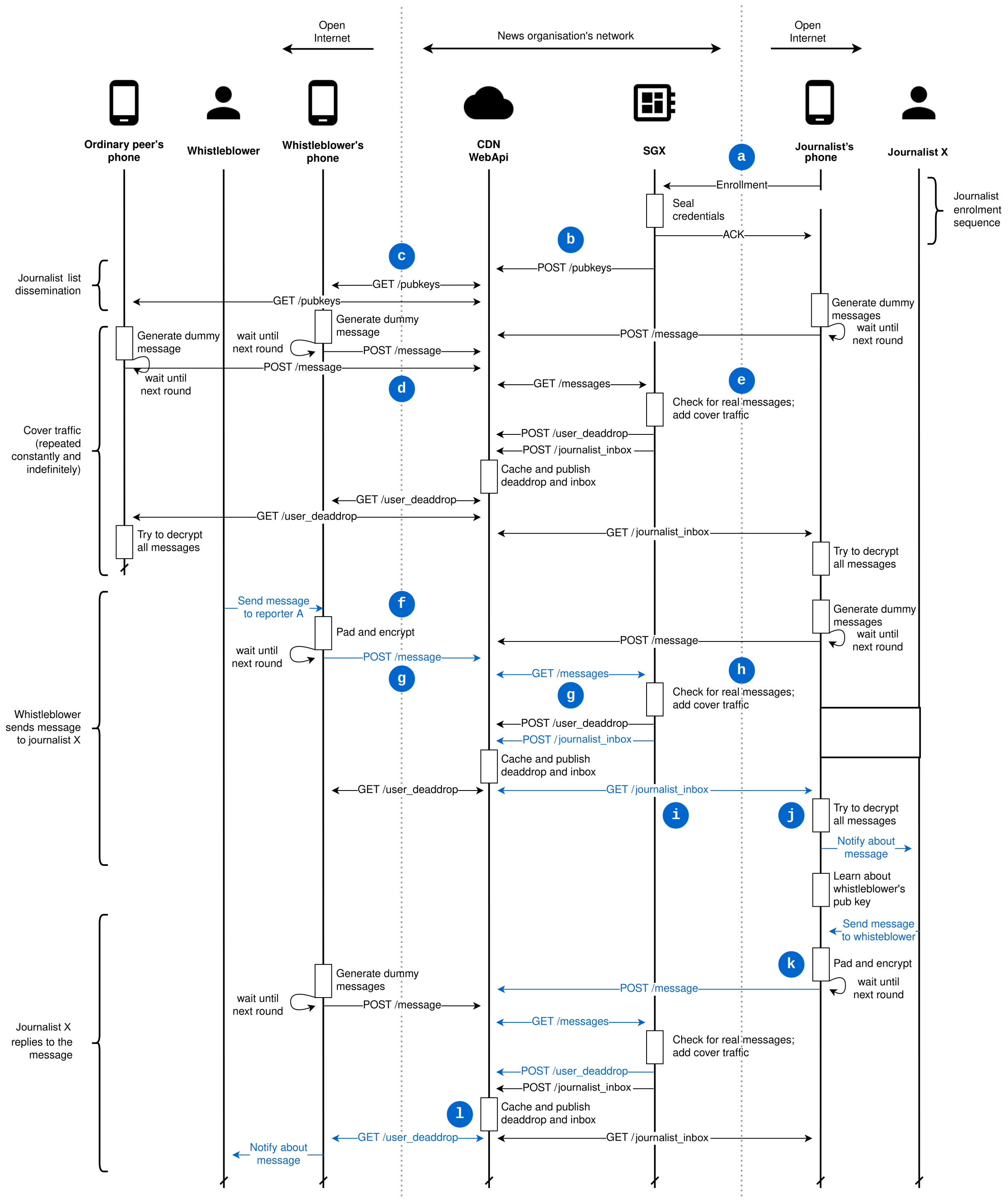


Figure 6.1: Protocol flow in CoverDrop.

There are two additional workflows for a journalist app: **retrieving messages** and **replying to a message**. For the former, the journalist app regularly fetches their inbox from the CDN and verifies the signature of each message ⓘ. Upon successful verification of the signature, the app then attempts to decrypt each message, which will only succeed for real messages ⓙ. The journalist now has access to the message contents, M_A, and can use pub_A to link the message to existing chats and when composing a response. For the latter, when the journalist replies to the source, their message M_J is encrypted with the receiver's public key and signed with the journalist's public key: $C_{J,inner} = E_{pub_A}(M_J, id_J, S_{priv_J}(M_J))$ ⓚ. This inner ciphertext is the encrypted again with the public key of the CoverNode's SGX enclave: $C_J = E_{pub_{SGX}}(C_{J,inner}, S_{priv_J}(C_{J,inner}))$. The message C_J is then sent to the CoverNode. Providing the signature for every message authored by a journalist ensures other users cannot impersonate a journalist with the scope of filling the dead drop or tricking users.

Similar to the journalist app, the source app also goes through an **enrolment** process. When any regular user starts the news reader app for the first time, the journalists' public key mappings `/pubkeys` are downloaded from the CDN ⓒ. The app creates an encrypted storage area on the device with a randomly chosen passphrase (see Section 6.2.3) and a buffer for outgoing messages, `out`.

There are three other flows for a source, two user-facing: **starting or retrieving a CoverDrop session**, and **contacting a journalist**; and the last refers to the message **dispatch process** which happens at regular intervals regardless of whether there is a message awaiting dispatch or not.

In order to send a secure message, a user either has to start a new CoverDrop session, or open an existing one. Every time they navigate to the CoverDrop screen inside the app, the user is presented with both options, regardless of whether a session already exists. If a passphrase is entered, CoverDrop uses $\mathrm{K}_{\mathrm{User}} + \mathrm{K}_{\mathrm{SE}}$ to try and decrypt the existing encrypted storage. If successful, all the relevant data, such as keys and message history, is retrieved from storage. Otherwise, fresh storage is created, pub_A, $priv_A$ stored, and then this is encrypted using the new passphrase $\mathrm{K}_{\mathrm{User}} + \mathrm{K}_{\mathrm{SE}}$. The fresh storage is not distinguishable from the one that was created when the app was started for the first time.

To contact a journalist by means of sending a message M_A, two layers of encryption are to be applied. The message M_A is padded to a fixed size and, together with the user's public key, is encrypted using the journalist's public key ⓕ: $C_{A,inner} = E_{pub_J}(pad(M_A, pub_A))$. This inner ciphertext is encrypted using the CoverNode's SGX public key: $C_A = E_{pub_{SGX}}(id_R, C_{A,inner})$. C_A is then added to the buffer of outgoing messages, *out*.

Last, messages are regularly sent to the CDN at every time interval `t`, where `t` is a configurable global parameter set by the news organisation. If there are any messages in the `out` buffer, then the oldest message is sent (using `POST /message` ⓓ), else a cover traffic message is created and sent using a dummy payload. Both types of messages have an outer layer of encryption, encrypted using the Cover Node's public key pub_{SGX}, which enables the detection and discarding of cover-traffic messages once decrypted within the SGX enclave ⓔ.

Lastly, the CoverNode handles traffic in two directions: **towards the news organisation**, and **towards news reader apps**.

The CoverNode receives encrypted cover traffic and messages from the CDN ⓖ. It attempts to decrypt the received message into $C_{A,inner}$. It awaits for enough messages to arrive, as well as creates any necessary dummy messages to fill the buffer ⓗ (see Section 6.2.3). The CoverNode signs every outgoing message and then publishes $(C_{A,inner}, S_{priv_{SGX}}(C_{A,inner}))$ to the journalist's inbox.

The CoverNode also handles the message traffic originating from within the news organisation towards users' news reader apps. When the CoverNode receives messages from the journalist side, it verifies the signature it receives and then posts the message to `user_deaddrop` ⓘ. Users' phones download the dead drops regularly and by trial and error they attempt decrypt the encrypted messages when the user has opened a CoverDrop session.

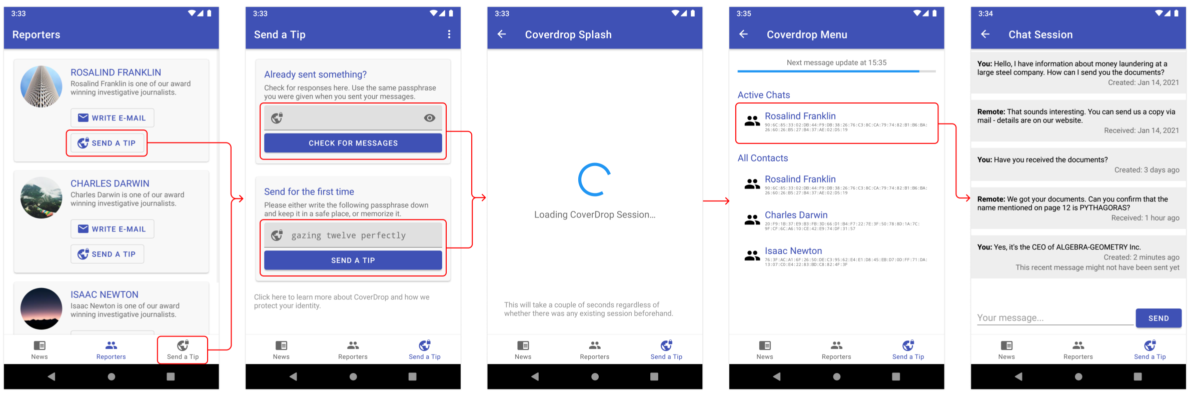
6.2.3 CoverDrop components

The are four primary components of CoverDrop: the CoverNode (sits in the news organisations infrastructure), the secure app library (provided to news organisation to include in their news reader app), a WebApi/CDN integration (which makes it easy to integrate the CoverDrop workflow within the news organisation networking infrastructure by leveraging their use of CDNs), and the user interface component (for which we make recommendations through our prototype, but leave to the news organisation's developers to customise). We discuss each of the components in turn.

The CoverNode is implemented as a Threshold Mix [191]. It manages traffic in two directions: forwarding messages from user devices to journalist devices, and in reverse. We describe the traffic from the user device to the journalist (reverse is similar). The CoverNode is a trusted proxy within the CoverDrop communications platform. It operates inside a secure enclave (in our prototype, SGX) to hide communication metadata from an outside observer that would connect a user with a journalist. It only communicates directly with the news organisation infrastructure (the journalist app) and with the CDN/WebApi. All the user apps send traffic towards the CoverNode (real or otherwise), but this is routed through the CDN/WebApi, thus protecting the CoverNode from DoS or timing attacks. The CoverNode polls the `/message` endpoint until enough messages have been received to meet the input threshold t_{in} (a configurable parameter). Each message is decrypted to learn $(id_R, C_{A,inner})$. The cover traffic messages are discarded and any real messages, $M_{J,actual}$, are collected for each journalist identified by id_J. The SGX enclave also generates enough cover traffic to meet the output threshold: $t_{out} - |M_{R,actual}|$. The cover traffic is also encrypted, but with a random key that does not match any journalist to ensure decryption for all journalists fails. All of the messages are signed by the SGX enclave and published to each journalist's inbox located on the CDN/WebApi. Observers only see that the CoverNode process/receive t_{in} and send t_{out} (both constant amounts) of fixed size encrypted messages, with $t_{in} \gg t_{out}$ because the majority of messages are cover traffic.

The CoverDrop app functionality can be integrated within existing news reader mobile applications with the help of **the secure app library**. Our prototype news reader app uses it to enable the CoverDrop functionality. A background service running at regular intervals ensures encrypted messages are sent to the news organisation. Every epoch, if a real message is not in the outgoing queue, then a cover message will be sent. By ensuring a constant output of fixed-size messages we therefore ensure that no information on whether communication is taking place is leaked. When the news reader app is open for the first time, it creates a fixed size (100 KiB) encrypted file and chooses a permanent salt value. We encrypt using a random passphrase pw and a fresh SE key K_{SE}. Creating this file regardless of whether a CoverDrop session exists ensures plausible deniability for the users of the news reader application. This file's "last-modified date" is updated every time the app is opened. We used cryptographic routines provided by the Android platform and the Argon2 password hash function. The process of starting a new CoverDrop session replaces the CoverDrop state with a newly encrypted empty one (padded to the fixed size of 100 KiB). The encryption process has three stages. First, the padded plaintext is encrypted using the `AES-GCM` encryption protocol with the key K_{SE} within the SE, returning the following ciphertext: $cipher'$. Second, using a Key Derivation Function (KDF) we derive K_{User} from pw and a salt value. And third, we encrypt $cipher'$ using `AES-CTR` with K_{User}, resulting in CoverDrop's encrypted state. Analogous to creating a new state for CoverDrop, a similar process is followed when a user accesses an existing CoverDrop state. Decryption of stored state depends on both the correct passphrase pw being entered, to enable correct derivation of K_{User}, as well as the existence of the correct K_{SE} within the SE. Whether pw is correct is not observable until the last step of decryption when the GCM tag is checked for validity. Therefore, only by passing $cipher'$ through the SE can an attacker verify passphrases. The user passphrase is generated from a wordlist, to ensure users can remember them easily. Using a wordlist also means that we can check for commonly mispelled words when the user enters a passphrase. This is an important validation which ensures that a common typo would not cause any existing CoverDrop session to be overwritten by starting a new session altogether. There are approximately $4.70 \cdot 10^{11}$ possible passphrases that can result from our list of 7777 words. Our experiments show that the bandwidth of the SE on a Pixel 3 device is limited to no more than 20 KiB/s, therefore limiting the efforts of a brute-force attack to 12 passphrases per minute. This is because of the requirement to pass $cipher'$ through the SE and it ensures it would take the attacker more than 70,000 years to try all possible combinations. This is based on the assumption that the SE is resistant to physical access and that attackers cannot extract any stored keys from the SE.

Ease of integration into existing infrastructure is central to our design, which we address


3:33
Reporters

ROSALIND FRANKLIN
Rosalind Franklin is one of our award winning investigative journalists.
WRITE E-MAIL
SEND A TIP

CHARLES DARWIN
Charles Darwin is one of our award winning investigative journalists.
WRITE E-MAIL
SEND A TIP

ISAAC NEWTON
Isaac Newton is one of our award winning investigative journalists.

News Reporters Send a Tip

3:33
Send a Tip

Already sent something?
Check for responses here. Use the same passphrase you were given when you sent your messages.
CHECK FOR MESSAGES

Send for the first time
Please either write the following passphrase down and keep it in a safe place, or memorize it.
gazing twelve perfectly
SEND A TIP

Click here to learn more about CoverDrop and how we protect your identity.

News Reporters Send a Tip

3:33
Coverdrop Splash

Loading CoverDrop Session...

This will take a couple of seconds regardless of whether there was any existing session beforehand.

News Reporters Send a Tip

3:35
Coverdrop Menu

Next message update at 15:35

Active Chats
Rosalind Franklin
90:6C:85:33:02:DB:44:F9:DB:38:26:76:C3:8C:CA:79:74:82:B1:B6:BA:26:60:26:B5:27:B4:37:AE:02:D5:19

All Contacts
Rosalind Franklin
90:6C:85:33:02:DB:44:F9:DB:38:26:76:C3:8C:CA:79:74:82:B1:B6:BA:26:60:26:B5:27:B4:37:AE:02:D5:19
Charles Darwin
20:F9:1B:37:E9:B3:FB:3D:66:D1:B4:F7:22:7E:3F:50:78:8D:1A:7C:9F:CF:6C:A6:10:CE:42:E9:74:DF:31:57
Isaac Newton
76:3F:AC:A1:6F:26:50:DE:C3:95:62:E4:E1:D8:45:EB:D7:0D:FF:71:DA:13:07:C0:E4:22:83:BD:C8:82:4F:3F

News Reporters Send a Tip

3:34
Chat Session

You: Hello, I have information about money laundering at a large steel company. How can I send you the documents?
Created: Jan 14, 2021

Remote: That sounds interesting. You can send us a copy via mail - details are on our website.
Received: Jan 14, 2021

You: Have you received the documents?
Created: 3 days ago

Remote: We got your documents. Can you confirm that the name mentioned on page 12 is PYTHAGORAS?
Received: 1 hour ago

You: Yes, it's the CEO of ALGEBRA-GEOMETRY Inc.
Created: 2 minutes ago
This recent message might not have been sent yet

Your message...
SEND

News Reporters Send a Tip

Figure 6.2: There are two entry points into CoverDrop: through journalist profiles and the main page. We always offer two options on login: recover a previous session *and* start a new session.

through two approaches: a **WebApi** to handle the POST requests and **ease of integration with existing CDNs** (see Chapter 5). As such, we facilitate CoverDrop's integration with existing CDNs, commonly used to distribute content and static resources, while also offering protections against DoS attacks. This approach lightens the load of the news organisation's web servers when it comes to handling cover traffic. The CDN maintains the journalist list and stores the dead drop for encrypted messages. All the clients are expected to access the dead drops at regular intervals. If clients miss some collection rounds (perhaps because their devices were offline) they can still access multiple old rounds because they are stored on the CDN for a set period (e.g. last 7 days). Clients are also expected to send a set amount of messages periodically (i.e. POST requests), which means that the rate-limiting feature of the CDN can disallow more requests than were expected. The WebApi web service handles all of the POST requests rather than sending them directly to the CoverNode. The WebApi protects the CoverNode (and in turn the TEE) from DoS and timing attacks, since it creates a message queue from all the POST requests it receives and allows the CoverNode to collect them on their own schedule.

Lastly, **the user interface** is a stand-alone implementation in order to demonstrate the type of interface a news organisation might develop. We did not include it inside the CoverDrop library to allow native integration within news reader apps, as well as facilitate branding and different workflows. Accessibility and minimal operational mistakes are vital to our UI design. Guided by our workshop responses (see Section 5.2), we place two entry points into CoverDrop: individual journalist's profiles and the main menu. The CoverDrop entry point always presents two options for any user who navigates to it (see Figure 6.2): starting a new CoverDrop session, or recovering a previous session. This is a deliberate to ensure an adversary with access to the user's phone learns nothing when loading the CoverDrop screen about existing sessions, and supported by the fact that CoverDrop retrieves its state only when the encrypted data is loaded into memory using the user's passphrase. Although the internal UI of CoverDrop looks like a messaging app to increase user familiarity, batching messages to benefit from the constant throughput channels means that messages incur delays (up to a maximum of the epoch time – set to one hour), which is not immediately obvious to a user. If we allowed messages to be sent immediately or exceed the cover traffic rate, it would be noticeable to a network adversary. We therefore, propose to explain these delays to users through a progress bar, explaining when the next batch of messages is dispatched. This is an implementation choice for our prototype, which is implemented by Daniel to demonstrate integration and facilitate performance measurements. The choice to include or adapt this feature is left to the news organisation's software development team.

6.2.4 Considerations for the multi-node proxy model

To simplify presentation, CoverDrop has been presented so far with only one proxy node that mixes the traffic, however it can support a multi-node model as well, in which multiple CoverNodes serially mix the traffic. These CoverNodes can either belong to the same news organisation, or they can belong to multiple collaborating news organisations, and we detail below the changes required for these approaches:

Same news organisation. The end device chooses a route through the mixes and onion-encrypts the message using the public keys of the SGX proxies. At each hop, the SGX enclave decrypts the outer layer of the message to find the public key of the next hop, then forwards the packet. Only at the final CoverNode it is revealed whether the message is real or cover. At this point the protocol behaves the same as the original specification (see Section 6.2).

Multiple news organisations. Different considerations need to be given to a multiple news organisation scenario, particularly load balancing since smaller news organisations may face difficulties coping with the larger amount of traffic commonly seen by larger news organisations. A form of load balancing based on the number of users could form the basis of a suitable approach, which we leave for future work. Further extensions to the work would be to allow cross-communication: users of news app A can contact a journalist at news organisation B.

6.3 Security analysis

In this section we provide a two-part security analysis: we review CoverDrop's security goals (see specification in Section 6.1.3) and analyse whether CoverDrop meets these with reference to an adversary that tries to defeat them; and we discuss what happens to the CoverDrop protections if our initial assumptions are violated (see Section 5.4.1). Lastly, we provide a brief discussion on CoverDrop's censorship resilience, an essential property for whistleblowing (see Section 5.2).

6.3.1 Security goals

We justify in this section why an attacker fails to defeat CoverDrop's security goals, provided that they operate within our adversarial model, and that the elements used in the CoverDrop infrastructure remain secure. We discuss what happens if these assumptions are violated in section 6.3.2.

Setup: The Adversary, *Adv*, monitors and records all communication on the public Internet and within the news organisation. *Adv* may also serve warrants to gain physical access to the CDN, CoverNode, and user devices. A whistleblower, *Alice*, wants to contact a journalist, *Bob*, by sending M_A. Adv may attack at the following times: T_a before Alice sends her initial message (Alice only sends regular cover traffic until this time); T_b during the communication (Message M_A is in-flight; Alice has an active CoverDrop session); T_c after the communication has taken place (Alice continues to send cover traffic and has an active CoverDrop session). At any point in time, T_a T_b T_c, we assume there are other app subscribers who opened the app recently, all of which send cover traffic messages.

G1: confidentiality & integrity. CoverDrop provides confidentiality and integrity of messages through authenticated encryption. The sender creates an ephemeral key pair for each message and the encryption layer therefore ensures all ciphertexts are independent.

G2: unobservable communication. As described in the setup, at times T_a and T_c Alice only sent cover messages. At time T_b Alice sent message M_A. We assume that *Adv* captures these messages between the app and the CoverDrop node, however, *Adv* cannot decide if the message M_A is real or cover because all the messages are the same size and encrypted for the CoverNode, which then decrypts them with its private key. Similar properties apply to all the messages the CoverNode sends to recipients. This suggests that no message in the CoverDrop system leaks any information about whether communication is taking place. Furthermore, the messages are regularly scheduled, so neither does the timing of messages leak any information. Therefore, the scheme achieves unobservable communication.

G3: plausible deniability. We identified three ways through which *Adv* might attempt to defeat the plausible deniability goal, namely whether *Adv* can tell that *Alice* has been using CoverDrop to communicate.

G3.A: *Adv* compromises the CDN. Regardless of the timing of the attack (T_a T_b T_c) the CDN is oblivious to the contents of the messages or final destination of the messages, hence compromising it does not give any advantage.

G3.B: *Adv* confiscates the CoverNode. Before communication T_a, all messages from Alice were for cover, so confiscating the CoverNode gives *Adv* no information. During communication T_b, the CoverNode has knowledge of legitimate messages versus cover messages. However, this information does not leave the SGX enclave and always resides in encrypted memory. Lastly, after delivery of the message T_c, the CoverNode will have deleted all previous messages. This happens after each epoch and if the system is powered down.

G3.C: *Adv* confiscates Alice's device. We briefly explain in Section 6.2.3 and in our publication [7] why the encrypted storage does not leak any information, even including brute-force attempts by the attacker. However, there are three types of attacks during T_b that *Adv* could perform: (i) If there is a message in the outgoing message queue, *Adv* can infer that Alice has recently composed a message that has not been sent yet;[2] (ii) If the CoverDrop session is unlocked and resides in the device's memory, *Adv* can examine all the session's state and potentially access sensitive information; (iii) If *Adv* confiscates Alice's device at two different times, *Adv* can tell if CoverDrop was used in the mean time. To counter this third attack, Alice can proactively manage the situation. When she becomes aware of *Adv*'s initial device access, she can publicly declare a new CoverDrop session with an unknown passphrase, effectively resetting it. This can be achieved, for example, by reinstalling the app and initiating a new (empty) session.

6.3.2 Compromised components

For our analysis of whether CoverDrop meets its proposed security goals in the face of a powerful adversary we assume the TEE, the SE and the third party libraries are not compromised. We discuss in this section the effect a compromise on either of these components has on the security protections CoverDrop provides.

Our prototype implementation uses Intel SGX to provide a Trusted Execution Environment (TEE) in which the CoverNode can decrypt the outer layer of the incoming messages in order to detect whether a message is real and requires forwarding or it needs to be discarded. The server on which a CoverNode runs is expected to be used for only this purpose, i.e. not to run any other applications on it. The SGX on the CoverNode is used to provide defense-in-depth by reducing the surface of possible attacks. Using SGX, however, does not completely remove the opportunity for attack, given recent side-channel attacks on the platform [27, 86, 216]. By defeating the security of the Intel SGX enclave, the adversary can distinguish between cover traffic and real messages. Although the message contents are still protected (only the outer layer of decryption is removed inside SGX), if the adversary has observed previous constant

[2]This attack can be countered by the app inserting cover messages into the message queue. However, this adds complexity and can lead to out-of-order delivery which has its own usability challenges. Therefore, we leave it for future work.

global network traffic, they can infer where the messages originated from, thus succeeding in performing a correlation attack. Two possible mitigations exist for the single CoverNode model to reduce the risk of compromise: either ensuring that the server is running the latest version of the SGX Platform Software, or that it has the newest hardware which already contains mitigations against speculative execution attacks and re-compiling client software with updated SDK [157]. An alternative approach would be to facilitate multiple CoverNodes in the news organisation's infrastructure (this also work across cooperative news organisations). In this situation, only by compromising all the nodes included in the message's path would the attacker learn the same information as in the single-node scenario. Just by compromising the entry or exit nodes (or even both), the attacker cannot correlate traffic to ensure the correlation attack succeeds. Therefore, this is a more robust and effective mitigation, particularly if the CoverNodes span multiple organisations.

We assume the SE on the user's device can resist physical attacks. Thus, the SE provides brute-force protection for the decryption passphrase. However, if this assumption is broken, it would enable an adversary to increase the rate of brute-force tries. This can be mitigated in-app by creating more complex passphrases to increase the difficulty to brute-force access. We suggest good practice is to suggest to users to delete sessions as soon as is practical by creating a new, empty session. However, doing so automatically might present usability issues, which we leave for future work.

It is common for mobile apps to rely on pre-built third-party libraries, however this makes it difficult for the news organisation to inspect the library for malicious behaviour, especially since most third party ad libraries already store and share personally-identifiable information with interested parties. We note positive movement towards in-house advertising [205], which enhances user protection.

6.3.3 Censorship resilience

Systems that facilitate secure communication for at-risk groups are often censored. We consider the type of censorship an attacker may attempt to perform for CoverDrop: DoS on the CoverNodes to disrupt the constant throughput channel, and banning news organisations that enable CoverDrop.

A DoS attack is a common way to subvert communication and deter users from using an application. To perform a DoS attack at the CoverNode level an attacker either creates an abundance of users which would overwhelm the SGX using cover traffic, or starts CoverDrop sessions and sends real messages towards the CoverNode (messages with no real purpose but to affect traffic through the news organisation's network). In the former, a suitable mitigation is to perform remote attestation with the SE, which means the attacker needs physical devices. It is much harder to prevent the latter since overfilling the real-message buffer (out) in the CoverNode results in dropping real messages. Having the CDN blocklist public keys or devices only affects small scale attackers, whereas powerful attackers such as nation states would not see a notable increase in cost. Unfortunately, spam is still an open problem for whistleblowing systems that do not have the ability to blocklist nodes (e.g. SecureDrop).

Placing blanket bans on news organisations or journalists is common in some countries where journalism is not offered full freedom, but even in other countries where freedom of the press is well established, there are still censored subjects [37, 211]. We see censorship on CoverDrop taking two possible paths. Either the attacker performs DoS on the CoverNode(s) (see discussion in previous paragraph), or attempts to stop news organisations from deploying/using CoverDrop. However, unlike the Tor model which has servers as endpoint in the network and an adversary can blocklist their individual IP addresses, CoverDrop does not expose servers outside the news organisation infrastructure making it a more difficult attack surface. To deter news organisations from using CoverDrop, the adversary has to censor the news organisation as a whole, an approach which depends on the freedom of the press in the relevant country or region.

6.4 Performance analysis

We used a prototype implementation to evaluate the performance of a news reader app using the CoverDrop functionality. We also evaluate the performance of our CoverNode implementation and the CDN/WebAPI integration.

6.4.1 Assumptions

We test our base case: one CoverNode proxy; 1 million users for the news app; 100 active CoverDrop sessions (for whistleblowing, ignoring those with non-sensitive comments and feedback); users send 5 real messages daily; 10 journalists signed up to receive CoverDrop messages; and journalists send 10 messages daily. These assumptions in the base case mean that with a set epoch of 1h, there are about 24 million cover messages daily and about 21 whistleblower messages per hour. The CoverNode's SGX enclave handles traffic in both directions: (1) for the user-to-journalist direction it awaits for 10^6 messages and outputs 10 messages (real or cover traffic), and (2) for the journalist-to-user it awaits for 100 messages and sends 10. In total 240 messages are delivered to the user dead-drop daily. We observe through measurements that traffic scales linearly with the number of CoverNodes, irrespective of organisation size.

6.4.2 News app

We implemented a sample news reader app and evaluated it using a Google Pixel 3 running Android 11. We looked at three main areas that would drive adoption down: on-device size requirements, battery consumption, and mobile data consumption. First, we observe that our prototype library implementation increases the size of the sample news app by less than 500 KB for most devices, which is significantly smaller than average news reader app sizes (usually more than 10 MB in size). Second, since we increase the amount of data usage with the constant throughput of the app, we compare the mobile data consumption of cover traffic messages (we consider that those users that send a real message would naturally expect an increase in data consumption) with that of network requests in our prototype news reader app. We observe that a CoverDrop message (real or otherwise since they are fixed size) uses about 6 KB including the HTTPS overhead, thus resulting in 4.3 MB monthly data consumption. This can be further reduced by using minimal protocol since the message payload is already encrypted, thus only using 360 KB per month. Furthermore, regular downloading of the dead-drop messages creates 110 KB in traffic per request (or 3.3 MB per month). This consumption is comparable to loading the home screen of a news app (1.5 MB) and browsing individual articles (typically larger than 100 KB). Third, we observe small impact on battery life, since CoverDrop's CPU and data usage is low compared to general news app usage. We conclude that the epoch could realistically be reduced without affecting battery consumption greatly, thus making CoverDrop also suitable for routine communication between journalists and readers.

Time delays are also of concern for a user-facing app such as the news reader app. As such, we looked at cover traffic message creation, which takes $7.5\mathrm{ms}$ ($\sigma = 1.4\mathrm{ms}$) of **CPU time** (less than $200\mathrm{ms}$ in total per day for an epoch of $1\mathrm{h}$), and message decryption $7.4\mathrm{ms}$ ($\sigma = 2.4\mathrm{ms}$), latter of which only applies to whistleblower's apps when there is an active CoverDrop session. Based on our assumptions above, the user's device downloads 240 messages daily from the dead-drop for each journalist from which they expect a message (here: 1), which leads to about 1.8 seconds of work when opening a session every 24 hours. The journalist's devices check significantly more messages daily (in our assumptions: $2,400$), however they only check against their own private key, leading to approximately 18 seconds of work per day.

Other areas of interest for the news app would be **API and hardware requirements** and ease-of-integration. Our implementation using the Hardware Security Element requires API level 29, which is only available to about 40% of the Android devices using the Google Play Store. Mitigations for lower API levels include increasing the password hashing parameters and not using a wordlist. App developers can tailor the user interface to match the already-existing app. To fully integrate the CoverDrop functionality they need to interact with the synchronous library calls and create four callbacks for interacting with the WebApi via HTTPS. For our prototype implementation, we used about 900 lines of Kotlin and 1100 lines of XML to implement the functionality above, including the UI functionality.

6.4.3 CoverNode

The CoverNode prototype is implemented in Intel SGX and consists of two components: the trusted enclave for message decryption and the untrusted application for handling network communication. Our performance measurements were conducted on a laptop with an i7-8565U processor (up to 4.6 GHz) decrypting messages (X25519 and XSalsa20) sent to the CoverNode. We observe a throughput of 833 messages per second (3 million messages per hour), considering batch processing of 1000 messages. Increasing the batch size decreases processing time by not having to transition between trusted and untrusted execution spaces. Increasing the number of enclave processes (dependent on the number of CPU cores) scales up performance and if this is not enough, we propose using multiple SGX servers and balancing the load using the CDN/WebAPI. We ensure through our protocol no data is stored to disk by the enclave, thereby not relying on the sealing functionality of SGX. Furthermore, we avoid using the monotonic counter functionality offered by SGX which has limited write cycles [139].

6.4.4 CDN/WebAPI

Lastly, we observe that CoverDrop's CDN integration is essential for news organisations. We calculate that with a 1 million user base, the CDN sees a total traffic volume of $102\mathrm{GB}$, which scales linearly with the number of users and fixed message throughput. Although in the wild these numbers are much reduced since the number of Daily Active Users (*DAU*) is much smaller than the number of users who installed an app, estimated at two orders of magnitude smaller [109], however getting accurate estimates for the former is non-trivial. We always consider our anonymity set relative to the DAU rather than the number of installs.

6.5 Comparison to existing systems

We compare CoverDrop with the most popular contact methods offered by news organisations (see Table 5.1). We split the comparison into four sections relevant across all contact methods: identifiers, resilience to warrants, correlation attacks, and usability.

Identifiers. CoverDrop is comparable with SecureDrop such that they both use system-generated random identifiers rather than leaking personally identifying information and all the traffic is encrypted, whereas the other systems either use personally identifiable information, such as a phone number, or do not encrypt traffic.

Warrants. Many of the individual components, including users, could have warrants served on them, as such resilience to warrants is an important feature. Although some of the contact methods, such as WhatsApp, Signal and physical mail fully support resilience to warrants on news organisations, since the news organisation wouldn't hold information relevant to identifying the whistleblower, they do not resist other types of warrants. We designed CoverDrop with a strong resilience to warrants in mind and thus, achieve the same resilience as SecureDrop.

	Features	Mail	Whatsapp	Signal	Email	PGP+Email	SecureDrop	CoverDrop
	No personally identifiable IDs	○	×	×	●	●	●	●
Identifiers	System generated random ID	NA	×	×	×	×	●	●
	Encrypts plaintext messages	×	●	●	×	●	●	●
	On third parties	×	×	○	×	○	●	●
	On news organisation	●	●	●	×	○	●	●
Resilience to warrants	On journalist	●	×	×	×	○	●	●
	On whistleblower post leak	●	×	×	○	○	●	●
	On whistleblower during leak	NA	×	×	○	○	●	●
	Resilience to global network adversary	○	×	×	×	×	○	●
Correlation attacks	Effective anonymity set at install	NA	●	○	●	○	×	●
	Effective anonymity set during leak	○	×	×	×	×	●	●
	Resilience to geo-location leakage	×	○	○	○	○	●	●
	Easy-to-use by a layperson	●	●	●	●	×	×	●
	Low latency two-way communication	×	●	●	●	●	×	○
Usability	Verifiable delivery of messages	×	●	●	×	×	×	●
	Spam prevention / rate limits	●	○	○	×	×	×	○
	High throughput	●	●	●	●	●	●	×

Table 6.1: Whistleblowing systems with fully (●), partially (○), and not supported (×) features.

Correlation attacks. Our strong baseline adversary can perform correlation attacks to identify whistleblowers. We design CoverDrop to withstand such an adversary, and thus, offer resilience to a global network adversary, and geo-location leakage. Furthermore, by design, CoverDrop relies on a constant throughput channel, a channel on which all the users send regular messages, whether real or not. Thus, the pre-existing user-base of the news application provides an effective anonymity set from CoverDrop's install. SecureDrop offers similarly strong properties, however, it suffers from a small effective anonymity set at install-time and building up an anonymity set is harder. The other systems do not offer resilience to correlation attacks, or only partially.

Usability. Lastly, CoverDrop attempts to bridge the gap between less secure means of communication (for the purpose of whistleblowing) that are easy to use and those with higher security guarantees that offer usability challenges for the layperson. By design CoverDrop looks like a common messaging app and is similarly easy-to-use like popular secure messaging apps, WhatsApp and Signal. Due to the delays incurred by the batching of messages, CoverDrop does not achieve the same low-latency two-way communication that email and contemporary secure messaging offer, however it does improve on latency when compared with SecureDrop. Unlike the other systems, CoverDrop does not offer high throughput, but it is suitable for secure initial contact.

We therefore find that CoverDrop is comparable in security with SecureDrop, making it a suitable lightweight and secure first contact option.

6.6 Summary

In this chapter we introduced CoverDrop, our proposed solution to secure the initial contact method used by sources when they want to communicate (establish trust) with journalists. CoverDrop's design has been guided by the main themes that emerged from our workshops with news organisation (see Chapter 5). It integrates within a news reader app using a secure library and works by creating a constant-throughput channel between the app installed on users' devices, therefore leveraging the large user base that news reader apps already have. Cover traffic flows through the existing CDN infrastructure used by the news organisation and real messages are only separated from cover traffic within the Secure Enclave of the CoverNode, implemented as a Threshold Mix. We evaluate the security of our approach and find that it can withstand a powerful global (and local) adversary. We also evaluate the performance of CoverDrop using a prototype implementation, including a sample news reader app, SGX implementation (the CoverNode), and the prototype CoverDrop secure library.

This chapter reviewed work that was published in the Proceedings of Privacy Enhancing Technologies Symposium 2022 [7]. The final design has been the result of contribution and feedback from all-co-authors: Mansoor, myself, Daniel, Ross and Alastair. My contributions are reflected in the final design, particularly ensuring system consistency with the feedback from our workshops (see Chapter 5), defining the security goals, and performing the security analysis. My colleagues contributed by implementing the SGX-based CoverNode (Mansoor), and the sample news app and CoverDrop mobile library (Daniel). Furthermore, Daniel measured the performance of the system and analysed the security properties offered by the encrypted storage on the mobile device through a semi-formal security analysis (found in Appendix A in the original publication [7]).

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

The aim of this work has been to understand and further secure encrypted communication. We started with the goal of detecting key server equivocation in end-to-end encrypted messaging systems. While this detection is commonly deferred to the user (Signal and WhatsApp through manual verification), or no transparency is provided at all (iMessage), we believe there is significant value in an automated trust-but-verify approach to trust establishment in secure messaging apps.

Chapter 3 began by investigating the type of key bindings commonly found in popular messaging systems, which we then formalised into Key-Name Graphs (KNGs). KNGs represent a generalised format for binding keys to human-readable names. We also introduced a new type of key insertion attack, which we called ghost-user attacks [217]. We address this new attack by proposing a baseline gossip protocol, which continuously monitors the KNGs provided by the key server to the users. The gossip protocol leverages a diversity of paths triggered by human mobility and is technology agnostic. Furthermore, the gossip protocol does not require any infrastructure changes to how the service provider stores the key-to-name bindings. The baseline gossip protocol targets co-location-based human friendship as a distribution means, however it can be extended to strangers, which we leave for future work. It's main contribution is that it can automatically differentiate between types of key changes to reduce the chance of false positive notifications without introducing any false negatives.

This baseline gossip protocol was analysed in Chapter 4. The analysis is two-fold: privacy and security analysis, and deployability analysis. We implemented the gossip protocol as a discrete event simulator to allow us to perform probabilistic model checking up to an upper bound of 14 700 friendship combinations between the users. We also analysed the protocol with respect to its desired core security properties: correctness, soundness, availability, and privacy. From this privacy and security analysis we learnt that one special type of key change is introduced by the gossip protocol which causes a false positive notification, and we propose mitigation. The deployability analysis helped us judge the effectiveness of the baseline protocol when we rely on the intersection between human physical mobility and online friendship. We conclude that such an approach is complementary only to an online key server and helps detect key server equivocation after-the-fact rather than replace a PKI's main utility. Automating these checks takes the burden away from the average user, whose threat model might not necessarily meet that of select user groups (such as whistleblowers, government agents, doctors, lawyers or others who deal with online confidential information), but who would nonetheless benefit from an alert when a ghost-user attack is performed on their account.

In Chapter 5 we aimed to better understand the security and privacy needs of one of these aforementioned select user groups, namely whistleblowers. This chapter presented an initial analysis of what systems are currently available for whistleblowers as offered by 24 worldwide newspapers, which are highly ranked on Alexa top sites. We learnt that only half of these offer an encrypted means of communication, although most of those use generic encrypted messaging apps with no form of metadata privacy. As follow-up, we then organised two workshops with several major British news organisations. These were performed in a semi-structured approach: we presented our idea, then had discussions triggered by both the participants as well as open-ended questions from the organisers. From these workshops we learnt that the area lacks a secure means of communication for sources to initiate contact to journalists. Sources are exposed to risk because often there is not a lot of emphasis placed on the security of the communication. The attending journalists provided insightful information about the source cultivation process, which typically changes in nature over time. This change may also trigger an increase in security need: it may start from an initial unhappiness with a situation, but evolve over time with the source blowing the whistle. Furthermore, we learnt that a good solution should balance usability with security, and a main driver of usability was identified as latency, since some of the more secure options encounter significant round-trip delays due to the enhanced security procedures. We analysed the workshop results and formalised a baseline adversary model to guide our solution to the problem.

The work is continued in Chapter 6 in which we presented CoverDrop, an initial contact system we designed to secure the initial contact method used by the sources who want to start communicating with a journalist. CoverDrop is included as a component inside the main news-reader app and creates a constant throughput channel between news reader's apps and a secure enclave inside the newspaper's infrastructure, the CoverNode, which then facilitates real traffic to journalists' devices by posting encrypted messages in dead-drops. CoverDrop was created to target some of the major problems that arose during our workshops: asynchronous communication, latency, usability and integration. Because it is designed as an initial contact option, the low bandwidth it offers is acceptable when balanced with the other priorities of low latency and enhanced metadata protection. Our security analysis showed that CoverDrop can withstand a powerful attacker by showing that CoverDrop meets its top security goals (confidentiality & integrity, unobservable communication and plausible deniability) with reference to our adversary model. From this analysis we initially excluded the possibility of compromised components and censorship, which we discuss separately. We concluded by analysing its performance through a prototype implementation of the newsreader app, the CoverNode (implemented in SGX) and the CoverDrop secure library.

Recently, I have been contributing to a collaborative effort with a team of software engineers at a major British news organisation in London to introduce CoverDrop into production and release it in their two newsreader apps on the iOS and Android platforms. Upon the completion of this process, the CoverDrop components will be open-sourced to simplify adoption by other news agencies. During the implementation stages, enhancements were made to the CoverDrop protocol, including the addition of future secrecy approaches, a key infrastructure, and a revised message (real or cover) sending strategy which avoids the restrictions imposed on apps running in the background in currently popular platforms, such as Android and iOS. These refinements will be detailed in a subsequent whitepaper and submitted for peer-review to an academic and industrial oversight board prior to release.

7.1 Future work

While the two solutions presented in this dissertation aim to improve both the transparency of trust establishment, and the security and privacy offered by modern secure communication apps, there are still areas to improve. We discuss below possible future work related to the problems tackled in this dissertation.

As discussed in Chapters 3 and 4, our solution for the baseline gossip protocol focuses on the connection between physical co-location and online friendship. While this presents opportunities to experiment with a gossip protocol that verifies the KNGs for friend's accounts, there are multiple extensions future research can investigate, which would target optimisations, for instance checking KNGs for those friends who do not meet in person regularly. One extension approach is to study is the reliability of peer-to-peer networks such as Dat [113], which we discuss in Section 3.5. Because the gossip protocol is technology agnostic we can remove the bounds of physical friend encounters. The immediate benefit of this approach is that it maintains the privacy nature of the baseline protocol to only verify with friends' accounts as opposed to gossiping with strangers.

Another possible research avenue is to explore gossiping beyond immediate friends by looking at friends-of-friends or even strangers. This complicates the gossip process both by increasing the burden on the battery since more comparisons will happen with nearby devices, and by requiring additional protection considerations.

Lastly, our gossip-based approach focuses on comparing users' KNGs in a pairwise manner, comparable with manual verification in which two users can verify they are communicating with the intended person. However, there is currently no mechanism to enable participants who belong to a group message to efficiently verify the KNGs of other group participants. Group messages present much of the same problems as two-participant chats, including being vulnerable to ghost-user attacks. The problem is that in group messages not all participants may know each other (as defined by having the other users' phone numbers in each other's contact lists), which the baseline gossip protocol does not tackle in its current form.

In Chapters 5 and 6 we worked to understand the needs of investigative journalists and their sources, and created a system to provide a secure initial contact method for sources. CoverDrop, as outlined in Chapter 6, is presented with several known limitations, which we discuss in Section 6.1.4. The biggest limitation that requires further investigation is the trade-off between providing cover traffic, which ensures the conversation is unobservable, and the low bandwidth provided by CoverDrop meant to keep data usage low for the regular users of the app. While higher bandwidth could support larger messages, or messages being sent more often, we expect this would have a direct and negative impact on the data usage patterns of regular users of the newsreader app.

Looking beyond the initial contact problem for journalists, future research could target to understand the requirements and identify the uniqueness of other areas related to journalism, such as long-term source cultivation, tip management, and secure collaboration between journalists. For instance, we learnt in our workshops that source cultivation can be a long process that most often involves changes in security needs because the source may evolve from being generally unhappy with a situation to blowing the whistle on illegality within their job. Our current solution, CoverDrop, is meant to secure the initial contact, but for a longer source cultivation process the journalist is expected to manage the transition to an alternative secure communication platform. This is an unreasonable expectation from the journalist and could be handled in a more automated way.

Furthermore, we need to understand and create suitable secure solutions to handle tip management once received on the news organisation side. This is because CoverDrop can be implemented to have both direct contact to investigative journalists and a more generic contact to particular "desks" (e.g. investigations desk, banking etc.) as a feature left to the news organisations choice. It is therefore expected that a manual triage will have to be done to filter out those tips that will not be followed up. From our workshops we learnt that a high degree of tips are actually not followed up, either due to human error or due to overwhelmingly high numbers of submissions.

Another important area that requires further research is secure collaboration. Journalists often collaborate with external contractors, or even other news agencies to investigate and build up a story. Such a collaborative effort often implies sharing sensitive information or even large document databases. To our knowledge there are no solutions targeting this space and it is up to individual journalists and the newspapers to manage these highly sensitive situations.

Looking beyond journalism, there are several other groups of users for whom standard end-to-end encrypted messaging is not enough. User groups such as lawyers and doctors would benefit from individualised approaches to security to meet the needs of their jobs.

BIBLIOGRAPHY

[1] Alfarez Abdul-Rahman. The PGP trust model. In *EDI-Forum: the Journal of Electronic Commerce*, volume 10, pages 27–31, 1997.

[2] Harold Abelson, Ross Anderson, Steven M Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G Neumann, et al. Keys Under Doormats: Mandating Insecurity by Requiring Government Access to All Data and Communications. *Journal of Cybersecurity*, 1(1):69–79, 2015.

[3] Utku Günay Acer and Otto Waltari. WiPush: Opportunistic notifications over WiFi without association. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, MobiQuitous 2017, page 353–362, New York, NY, USA, 2017. Association for Computing Machinery.

[4] Mohammad Afaneh. How to achieve ranges of over 1 km using Bluetooth Low Energy. `https://novelbits.io/bluetooth-long-range-how-to-achieve-results-over-1km/`, 2020. `https://perma.cc/U3F6-H5X8`.

[5] Arun Kumar Agrawal and Sanchit Mehrotra. Application of Elliptic Curve Cryptography in Pretty Good Privacy (PGP). In *2016 International Conference on Computing, Communication and Automation (ICCCA)*, pages 924–929. IEEE, 2016.

[6] Mansoor Ahmed-Rengers, Ilia Shumailov, and Ross Anderson. Snitches Get Stitches: On The Difficulty Of Whistleblowing. In *Proceedings of the 27th International Workshop on Security Protocols*, 2019.

[7] Mansoor Ahmed-Rengers, Diana A Vasile, Daniel Hugenroth, Alastair R Beresford, and Ross Anderson. CoverDrop: Blowing the Whistle Through A News App. *Proceedings on Privacy Enhancing Technologies*, 2022(2):47–67, 2022.

[8] Mustafa Al-Bassam and Sarah Meiklejohn. Contour: A practical system for Binary Transparency. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 94–110. Springer, 2018.

[9] Chris Alexander and Ian Goldberg. Improved User Authentication in Off-the-Record Messaging. In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, pages 41–47, 2007.

[10] Device Analyzer. Key-value pairs, 2017. `https://perma.cc/JN7J-8KEG`.

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

[12] Apple. Apple Platform Security - iMessage overview. `https://support.apple.com/en-gb/guide/security/secd9764312f/1/web/1`, 2020. `https://perma.cc/5EBJ-XW44`.

[13] Apple Inc. Apple reports first quarter results. `https://www.apple.com/newsroom/2018/02/apple-reports-first-quarter-results`, February 2018. `https://perma.cc/M6WV-Q4HK`.

[14] National Archives. Pentagon papers, 2019.

[15] Valerio Arnaboldi, Andrea Passarella, Marco Conti, and Robin IM Dunbar. *Online social networks: human cognitive constraints in Facebook and Twitter personal graphs*. Elsevier, 2015.

[16] ArsTechnica. Have a confidential news tip for Ars Technica? `https://arstechnica.com/news-tips/`, 2019. `https://perma.cc/D4ZG-4FME`.

[17] Dirk Balfanz, Glenn Durfee, Diana K Smetters, and Rebecca E Grinter. In search of usable security: Five lessons from the field. *IEEE Security & Privacy*, 2(5):19–24, 2004.

[18] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. ARPKI: Attack resilient public-key infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 382–393, 2014.

[19] Matthias Bauer. New covert channels in HTTP: adding unwitting Web browsers to anonymity sets. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, pages 72–78, 2003.

[20] Jiang Bian, Remzi Seker, and Umit Topaloglu. Off-the-record instant messaging for group conversation. In *2007 IEEE International Conference on Information Reuse and Integration*, pages 79–84. IEEE, 2007.

[21] Kemal Bicakci, Enes Altuncu, Muhammet Sakir Sahkulubey, Hakan Ezgi Kiziloz, and Yusuf Uzunay. How safe is safety number? A user study on Signal's fingerprint and safety number methods for public key verification. In *International Conference on Information Security*, pages 85–98. Springer, 2018.

[22] Ken Birman. The promise, and limitations, of gossip protocols. *ACM SIGOPS Operating Systems Review*, 41(5):8–13, 2007.

[23] Archie Bland. FireChat – the messaging app that's powering the Hong Kong protests, September 2014. `https://www.theguardian.com/world/2014/sep/29/firechat-messaging-app-powering-hong-kong-protests`.

[24] Archie Bland. The Latest Chat App for iPhone Needs No Internet Connection, March 2014. `https://web.archive.org/web/20151127150930/http://www.technologyreview.com/news/525921/the-latest-chat-app-for-iphone-needs-no-internet-connection/`.

[25] Joseph Bonneau. EthIKS: Using Ethereum to audit a CONIKS key transparency log. In *International Conference on Financial Cryptography and Data Security*, pages 95–105. Springer, 2016.

[26] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, pages 77–84, 2004.

[27] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software Grand Exposure: SGX Cache Attacks Are Practical. *arXiv e-prints*, page arXiv:1702.07521, February 2017.

[28] Cabal. Experimental p2p community chat platform, January 2019. `https://cabal.chat`.

[29] Deanna D Caputo, Shari Lawrence Pfleeger, M Angela Sasse, Paul Ammann, Jeff Offutt, and Lin Deng. Barriers to usable security? Three organizational case studies. *IEEE Security & Privacy*, 14(5):22–32, 2016.

[30] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. Seemless: Secure end-to-end encrypted messaging with less trust. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1639–1656, 2019.

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

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

[33] Peng Cheng, Ibrahim Ethem Bagci, Utz Roedig, and Jeff Yan. Sonarsnoop: Active acoustic side-channel attacks. *International Journal of Information Security*, 19:213–228, 2020.

[34] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 415–423. IEEE, 2015.

[35] Mike Clark. How we combat scraping. `https://about.fb.com/news/2021/04/how-we-combat-scraping/`, 2021. `https://perma.cc/2WCA-5QFL`.

[36] CNN. Tips. `https://edition.cnn.com/tips/`.

[37] The Conversation. Al-jazeera trial and ukraine abduction set dark tone for world press freedom day. `https://theconversation.com/al-jazeera-trial-and-ukraine-abduction-set-dark-tone-for-world-press-freedom-day-25946`, 2022. `https://perma.cc/6JHM-KPL7`.

[38] Henry Corrigan-Gibbs. Keeping secrets. `https://stanfordmag.org/contents/keeping-secrets`, 2014. `https://perma.cc/6CZU-GMMF`.

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

[40] Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 340–350, 2010.

[41] Henry Corrigan-Gibbs and Bryan Ford. Conscript your friends into larger anonymity sets with JavaScript. In *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society*, pages 243–248, 2013.

[42] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in Verdict. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 147–162, 2013.

[43] Sergiu Costea, Marios O Choudary, Doru Gucea, Björn Tackmann, and Costin Raiciu. Secure opportunistic multipath key exchange. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2077–2094, 2018.

[44] Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster protocol version 2< draft-moeller-v2-01.txt. *Online specification*, 2003.

[45] Justin Cranshaw, Eran Toch, Jason Hong, Aniket Kittur, and Norman Sadeh. Bridging the gap between physical location and online social networks. *Proceedings of the 12th ACM International Conference on Ubiquitous Computing (Ubicomp)*, page 119, 2010.

[46] Joan Daemen and Vincent Rijmen. The Advanced Encryption Standard Process. In *The Design of Rijndael*, pages 1–8. Springer, 2002.

[47] China Daily. Contact us. `http://www.chinadaily.com.cn/e/static_e/contact`, 2019. `https://perma.cc/89DY-BBT2`.

[48] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *2003 Symposium on Security and Privacy, 2003.*, pages 2–15. IEEE, 2003.

[49] R Davis. The data encryption standard in perspective. *IEEE Communications Society Magazine*, 16(6):5–9, 1978.

[50] Dawn. Contact us. `https://www.dawn.com/contact/`, 2019. `https://perma.cc/3UD9-PKN9`.

[51] Apple Developer. Bonjour, 2021. `https://developer.apple.com/bonjour/`.

[52] Google Developers. Pixel Binary Transparency Technical Details Page. `https://developers.google.com/android/binary_transparency/pixel`, 2022. `https://perma.cc/N7RQ-22G7`.

[53] Prithula Dhungel, Moritz Steiner, Ivinko Rimac, Volker Hilt, and Keith W Ross. Waiting for anonymity: Understanding delays in the Tor overlay. In *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*, pages 1–4. IEEE, 2010.

[54] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Secure off-the-record messaging. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, pages 81–89, 2005.

[55] Philip Di Salvo. Securing whistleblowing in the digital age: Securedrop and the changing journalistic practices for source protection. *Digital Journalism*, 9(4):443–460, 2021.

[56] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

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

[58] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure logging schemes and certificate transparency. In *European Symposium on Research in Computer Security*, pages 140–158. Springer, 2016.

[59] Paul Ducklin. The TURKTRUST SSL certificate fiasco – what really happened, and what happens next? `https://nakedsecurity.sophos.com/2013/01/08/the-turktrust-ssl-certificate-fiasco-what-happened-and-what-happens-next/`, 2013. `https://perma.cc/32XK-339U`.

[60] Robin Dunbar. How many "friends" can you really have? *IEEE Spectrum*, 48(6), 2011.

[61] Robin IM Dunbar. Do online social media cut through the constraints that limit the size of offline social networks? *Royal Society Open Science*, 3(1):150292, 2016.

[62] Robin IM Dunbar, Valerio Arnaboldi, Marco Conti, and Andrea Passarella. The structure of online social networks mirrors those in the offline world. *Social networks*, 43:39–47, 2015.

[63] Steven Dusse, Paul Hoffman, Blake Ramsdell, Laurence Lundblade, and Lisa Repka. S/MIME version 2 message specification. *IETF RFC*, 2311, 1998.

[64] Morris J Dworkin, Elaine B Barker, James R Nechvatal, James Foti, Lawrence E Bassham, E Roback, James F Dray Jr, et al. Advanced Encryption Standard (AES). 2001.

[65] Nathan Eagle, Alex Sandy Pentland, and David Lazer. Inferring friendship network structure by using mobile phone data. *Proceedings of the National Academy of Sciences*, 106(36):15274–15278, 2009.

[66] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI certificate theory. Technical report, RFC 2693, 1999.

[67] Saba Eskandarian, Eran Messeri, Joseph Bonneau, and Dan Boneh. Certificate transparency with privacy. *arXiv preprint arXiv:1703.02209*, 2017.

[68] Private Eye. Contact, 2019. `https://www.private-eye.co.uk/about/contact`.

[69] Ellis Fenske, Dane Brown, Jeremy Martin, Travis Mayberry, Peter Ryan, and Erik C Rye. Three years later: A study of mac address randomization in mobile devices and when it succeeds. *Proc. Priv. Enhancing Technol.*, 2021(3):164–181, 2021.

[70] Mahdi Daghmehchi Firoozjaei, Sang Min Lee, and Hyoungshick Kim. O^{2}TR: Offline Off-the-Record (OTR) Messaging. In *International Workshop on Information Security Applications*, pages 61–71. Springer, 2017.

[71] The Matrix.org Foundation. Matrix specification, November 2023. `https://spec.matrix.org/latest/`.

[72] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL Protocol Version 3.0. Internet-Draft draft-ietf-tls-ssl-version3-00, Internet Engineering Task Force, November 1996. Work in Progress.

[73] Kevin Gallagher, Sameer Patil, and Nasir Memon. New Me: Understanding Expert and {Non-Expert} Perceptions and Usage of the Tor Anonymity Network. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 385–398, 2017.

[74] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly Media, Inc., 1995.

[75] Simson Garfinkel and Heather Richter Lipford. Usable security: History, themes, and challenges. *Synthesis Lectures on Information Security, Privacy, and Trust*, 5(2):1–124, 2014.

[76] Simson L Garfinkel and Robert C Miller. Johnny 2: A user test of key continuity management with S/MIME and Outlook Express. In *Proceedings of the 2005 symposium on Usable privacy and security*, pages 13–24, 2005.

[77] Oliver Gasser, Benjamin Hof, Max Helm, Maciej Korczynski, Ralph Holz, and Georg Carle. In log we trust: Revealing poor security practices with certificate transparency logs and internet measurements. In *International Conference on Passive and Active Network Measurement*, pages 173–185. Springer, 2018.

[78] Alessandro Gattolin, Cristina Rottondi, and Giacomo Verticale. Blast: Blockchain-assisted key transparency for device authentication. In *International Forum on Research and Technology for Society and Industry (RTSI)*, pages 1–6. IEEE, 2018.

[79] The Globe and Mail. PGP directory and SecureDrop links, 2018. PGP directory (`https://sec.theglobeandmail.com/pgp/`) and SecureDrop (`https://sec.theglobeandmail.com/securedrop/`).

[80] O Globo. Contact us (Portuguese), 2019. `https://oglobo.globo.com/fale-conosco/`.

[81] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.

[82] Ian Goldberg, Berkant Ustaoğlu, Matthew D Van Gundy, and Hao Chen. Multi-party off-the-record messaging. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 358–368, 2009.

[83] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

[84] Philippe Golle and Ari Juels. Dining cryptographers revisited. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 456–473. Springer, 2004.

[85] Bruno Gonçalves, Nicola Perra, and Alessandro Vespignani. Modeling users' activity on twitter networks: Validation of Dunbar's number. *PloS one*, 6(8), 2011.

[86] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache Attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security*, EuroSec'17, New York, NY, USA, 2017. Association for Computing Machinery.

[87] Matthew Green and Matthew Smith. Developers are not the enemy!: The need for usable security apis. *IEEE Security & Privacy*, 14(5):40–46, 2016.

[88] Joel Greenberg. The Enigma machine. In *The Turing Guide*. Oxford University Press, 2017.

[89] NCC Group. Keybase protocol security review, 2018.

[90] The Guardian. The NSA Files. `https://www.theguardian.com/world/interactive/2013/nov/01/snowden-nsa-files-surveillance-revelations-decoded`, 2013. `https://perma.cc/P66G-K5HH`.

[91] The Guardian. How to contact the Guardian securely. `https://www.theguardian.com/help/ng-interactive/2017/mar/17/contact-the-guardian-securely`, 2017. `https://perma.cc/Q48M-QQP6`.

[92] Thomas C Hales. The NSA back door to NIST. *Notices of the AMS*, 61(2):190–192, 2013.

[93] Luke Harding. *The Snowden files: The inside story of the world's most wanted man*. Guardian Faber Publishing, 2014.

[94] The Sydney Morning Herald. Contact us, 2019. `https://www.smh.com.au/contact-us`.

[95] Amir Herzberg, Hemi Leibowitz, Kent Seamons, Elham Vaziripour, Justin Wu, and Daniel Zappala. Secure messaging authentication ceremonies are broken. *IEEE Security & Privacy*, 19(2):29–37, 2020.

[96] Matthew Hodgson. Introducing p2p matrix, June 2020. `https://matrix.org/blog/2020/06/02/introducing-p2p-matrix/`.

[97] B. Hof. STH Cross Logging. draft-hof-trans-cross-00, 2018.

[98] Nicholas Hopper, Eugene Y Vasserman, and Eric Chan-Tin. How much anonymity does network latency leak? *ACM Transactions on Information and System Security (TISSEC)*, 13(2):1–28, 2010.

[99] Chatham House. Chatham house rule. https://www.chathamhouse.org/about-us/chatham-house-rule.

[100] Joel Hruska. The NSA wants 'front door' access to your encrypted data. https://www.extremetech.com/defense/203275-the-nsa-wants-front-door-access-to-your-encrypted-data, 2015. https://perma.cc/Z8G4-FKJZ.

[101] Apple Inc. and Google. Privacy-preserving contact tracing, 2020. https://www.apple.com/covid19/contacttracing.

[102] The Intercept. The Intercept welcomes whistleblowers, 2020. https://theintercept.com/source/.

[103] Amnesty International. Forensic Methodology Report: How to catch NSO Group's Pegasus. https://www.amnesty.org/en/latest/research/2021/07/forensic-methodology-report-how-to-catch-nso-groups-pegasus/, 2021.

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

[105] Craig Jarvis. *Crypto Wars: The Fight for Privacy in the Digital Age: a Political History of Digital Encryption*. CRC Press, 2020.

[106] H. Jayakrishnan and R. Murali. A Simple and Robust End-to-End Encryption Architecture for Anonymous and Secure Whistleblowing. In *2019 Twelfth International Conference on Contemporary Computing (IC3)*, pages 1–6, 2019.

[107] Kee Jefferys, Maxim Shishmarev, and Simon Harman. Session: A Model for End-To-End Encrypted Conversations With Minimal Metadata Leakage. *arXiv preprint arXiv:2002.04609*, 2020.

[108] Andrey Jivsov. Elliptic curve cryptography (ECC) in OpenPGP. Technical report, 2012.

[109] Joseph Johnson. Daily active users (DAU) of leading iPhone news apps in the United Kingdom (UK) during October 2020, 2020. https://www.statista.com/statistics/878573/leading-iphone-news-apps-dau-united-kingdom/.

[110] Wall Street Journal. Contact us, 2019. `https://customercenter.wsj.com/contact`.

[111] David Kahn. *The Codebreakers: The comprehensive history of secret communication from ancient times to the internet.* Simon and Schuster, 1996.

[112] Dimitris N Kalofonos, Zoe Antoniou, Franklin D Reynolds, Max Van-Kleek, Jacob Strauss, and Paul Wisner. Mynet: A platform for secure p2p personal and social networking services. In *International Conference on Pervasive Computing and Communications (PerCom)*, pages 135–146. IEEE, 2008.

[113] Duncan Keall. How dat works, January 2019. `https://perma.cc/GU7Y-LFB5`.

[114] Rohit Khare and Adam Rifkin. Weaving a web of trust. *World Wide Web Journal*, 2(3):77–112, 1997.

[115] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil Gligor. Accountable key infrastructure (AKI) a proposal for a public-key validation infrastructure. In *Proceedings of the 22nd international conference on World Wide Web*, pages 679–690, 2013.

[116] Martin Kleppmann, Stephan A Kollmann, Diana A Vasile, and Alastair R Beresford. From secure messaging to secure collaboration. In *Cambridge International Workshop on Security Protocols*, pages 179–185. Springer, 2018.

[117] P Kocker, A Freier, and P Karlton. The ssl protocol version 3.0. *Netscape Communications Corp*, 1996.

[118] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security '16)*, pages 279–296, 2016.

[119] Brian Kondracki, Johnny So, and Nick Nikiforakis. Uninvited guests: Analyzing the identity and behavior of certificate transparency bots. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[120] Nikita Korzhitskii and Niklas Carlsson. Characterizing the root landscape of certificate transparency logs. In *2020 IFIP Networking Conference (Networking)*, pages 190–198. IEEE, 2020.

[121] Murat Yasin Kubilay, Mehmet Sabir Kiraz, and Hacı Ali Mantar. CertLedger: A new PKI model with Certificate Transparency based on blockchain. *Computers & Security*, 85:333–352, 2019.

[122] Karthik Lakshminarayanan and Erika Trautman. Google workspace delivers new levels of trusted collaboration for a hybrid work world, 2021. https://perma.cc/V9N9-5YCG.

[123] Butler Lampson. Privacy and security usable security: how to get it. *Communications of the ACM*, 52(11):25–27, 2009.

[124] Susan Landau. Making sense from Snowden: What's significant in the NSA surveillance revelations. *IEEE Security & Privacy*, 11(4):54–63, 2013.

[125] Susan Landau. The second crypto {War—What's} different now. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[126] Paul Lashmar. No more sources? The impact of Snowden's revelations on journalists and their confidential sources. *Journalism Practice*, 11(6):665–688, 2017.

[127] Ben Laurie. Certificate transparency. *ACM Queue*, 12(8):10, 2014.

[128] Sean Lawlor and Kevin Lewi. Deploying key transparency at WhatsApp. https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/, 2023. https://perma.cc/6GZE-LBTQ.

[129] Joao Leitao, José Pereira, and LuÍs Rodrigues. Gossip-based broadcast. In *Handbook of Peer-to-Peer Networking*, pages 831–860. Springer, 2010.

[130] Ada Lerner, Eric Zeng, and Franziska Roesner. Confidante: Usable encrypted email: A case study with lawyers and journalists. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 385–400. IEEE, 2017.

[131] Ian Levy and Crispin Robinson. Principles for a more informed exceptional access debate. https://www.lawfareblog.com/principles-more-informed-exceptional-access-debate, November 2018. https://perma.cc/7RJK-FM32.

[132] Bingyu Li, Jingqiang Lin, Fengjun Li, Qiongxiao Wang, Qi Li, Jiwu Jing, and Congli Wang. Certificate transparency in the wild: Exploring the reliability of monitors. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2505–2520, 2019.

[133] Hong Liu, Eugene Y Vasserman, and Nicholas Hopper. Improved group off-the-record messaging. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, pages 249–254, 2013.

[134] Susan Lund, Anu Madgavkar, James Manyika, and Sven Smit. What's next for remote work: An analysis of 2 000 tasks, 800 jobs, and nine countries, 2020. https://perma.cc/9JQR-YFF5.

[135] The Mainichi. Contact form, 2019. `https://form.mainichi.jp/mdn/common/content.html`.

[136] Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghosh, Ercan Oztürk, Kevin Lewi, and Sean Lawlor. Parakeet: Practical key transparency for end-to-end encrypted messaging. *Cryptology ePrint Archive*, 2023.

[137] Moxie Marlinspike and Trevor Perrin. The Sesame Algorithm: Session Management for Asynchronous Message Encryption. `https://signal.org/docs/specifications/sesame/`, 2017. `https://perma.cc/NDZ3-FWC2`.

[138] Jeremy Martin, Travis Mayberry, Collin Donahue, Lucas Foppe, Lamont Brown, Chadwick Riggins, Erik C Rye, and Dane Brown. A study of MAC address randomization in mobile devices and when it fails. *Proceedings on Privacy Enhancing Technologies*, 2017(4):365–383, 2017.

[139] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. ROTE: Rollback protection for trusted execution. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1289–1306, Vancouver, BC, August 2017. USENIX Association.

[140] Célestin Matte, Mathieu Cunche, Franck Rousseau, and Mathy Vanhoef. Defeating MAC address randomization through timing attacks. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 15–20, 2016.

[141] Ueli Maurer. Modelling a public-key infrastructure. In *European Symposium on Research in Computer Security*, pages 325–350. Springer, 1996.

[142] Susan E McGregor, Polina Charters, Tobin Holliday, and Franziska Roesner. Investigating the computer security practices and needs of journalists. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 399–414, 2015.

[143] Susan E McGregor, Elizabeth Anne Watkins, Mahdi Nasrullah Al-Ameen, Kelly Caine, and Franziska Roesner. When the weakest link is strong: Secure collaboration in the case of the Panama Papers. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 505–522, 2017.

[144] Sarah Meiklejohn, Joe DeBlasio, Devon O'Brien, Chris Thompson, Kevin Yeo, and Emily Stark. SoK: SCT Auditing in Certificate Transparency. *arXiv preprint arXiv:2203.01661*, 2022.

[145] Marcela Melara. Why making Johnny's key management transparent is so challenging. `https://freedom-to-tinker.com/2016/03/31/why-making-johnny`

s-key-management-transparent-is-so-challenging/, March 2016. https://perma.cc/RX2S-MZQH.

[146] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: Bringing key transparency to end users. In *USENIX Security Symposium*, pages 383–398, 2015.

[147] Joseph Menn. Key Internet operator VeriSign hit by hackers. https://www.reuters.com/article/us-hacking-verisign-idUSTRE8110Z820120202, 2012. https://perma.cc/45AS-LSEK.

[148] Judith Miller. Why confidential sources are important and why I would go to jail to protect them. *Comm. Law.*, 22:4, 2004.

[149] Le Monde. Contact the editor (French), 2019. https://www.lemonde.fr/faq/?question=28465-contacter-redaction-28465.

[150] Mozilla. Security/ Binary Transparency. https://wiki.mozilla.org/Security/Binary_Transparency, 2017.

[151] Francesca Musiani and Ksenia Ermoshina. What is a good secure messaging tool? The EFF secure messaging scorecard and the shaping of digital (usable) security. *Westminster Papers in Communication and Culture*, 12(3), 2017.

[152] BBC News. How to share your questions, stories, pictures and videos with BBC News, 2018. https://www.bbc.co.uk/news/10725415.

[153] BBC News. 'Whistleblower' taped to chair and gagged, 2018. https://www.bbc.co.uk/news/uk-scotland-44222575.

[154] BuzzFeed News. Share tips securely & anonymously, 2018. https://contact.buzzfeed.com/?country=en-uk.

[155] Hoang-Long Nguyen, Claudia-Lavinia Ignat, and Olivier Perrin. Trusternity: Auditing transparent log server with blockchain. In *Companion Proceedings of the The Web Conference 2018*, pages 79–80, 2018.

[156] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. {CHAINIAC}: Proactive {Software-Update} transparency via collectively signed skipchains and verified builds. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1271–1287, 2017.

[157] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. A Survey of Published Attacks on Intel SGX, 2020.

[158] Greg Norcie, Jim Blythe, Kelly Caine, and L Jean Camp. Why Johnny can't blow the whistle: Identifying and reducing usability issues in anonymity systems. In *Workshop on Usable Security*, volume 6, pages 50–60, 2014.

[159] L. Nordberg, D. Gillmor, and T. Ritter. Gossiping in CT. internet-draft draft-ietf-trans-gossip-05, 2018.

[160] NYM. Building the next generation of privacy infrastructure, 2023. `https://nymtech.net`.

[161] U.S Department of Homeland Security. The Menlo Report, ethical principles guiding information and communication technology research. `https://www.dhs.gov/sites/default/files/publications/CSD-MenloPrinciplesCORE-20120803_1.pdf`, 2012. `https://perma.cc/GD26-SMC9`.

[162] The Times of India. Main page, 2019. `https://timesofindia.indiatimes.com`.

[163] Cellular Operators Association of India (COAI). Mobile subscriber figure report, 2012. `https://www.coai.com/statistics/gsm-subscriber-figure-report`.

[164] Spiegel Online. How to contact the Spiegel (German). `https://www.spiegel.de/extra/so-nehmen-informanten-sicheren-kontakt-zum-spiegel-auf-a-1030502.html`, 2019. `https://perma.cc/DW8E-C8MQ`.

[165] Oxen. Oxen service nodes. `https://docs.oxen.io/about-the-oxen-blockchain/oxen-service-nodes`.

[166] El Pais. Contact us (Spanish), 2019. `https://elpais.com/estaticos/contacte/`.

[167] Andriy Panchenko, Fabian Lanze, and Thomas Engel. Improving performance and anonymity in the Tor network. In *2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC)*, pages 1–10. IEEE, 2012.

[168] Vandana Pednekar-Magal and Peter Shields. The state and telecom surveillance policy: The clipper chip initiative. *Communication Law and Policy*, 8(4):429–464, 2003.

[169] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The Loopix Anonymity System. *CoRR*, abs/1703.00536, 2017.

[170] Tim Polk and Sean Turner. Prohibiting Secure Sockets Layer (SSL) Version 2.0. RFC 6176, March 2011.

[171] The Washington Post. Send a letter to the editor, 2019. `https://helpcenter.washingtonpost.com/hc/en-us/articles/236004788-Send-a-letter-to-the-editor`.

[172] ProPublica. NY Fed Fired Examiner Who Took on Goldman, 2013. `https://www.propublica.org/article/ny-fed-fired-examiner-who-took-on-goldman`.

[173] Jian Qiao, Xuemin Sherman Shen, Jon W Mark, Qinghua Shen, Yejun He, and Lei Lei. Enabling device-to-device communications in millimeter-wave 5G cellular networks. *IEEE Communications Magazine*, 53(1):209–215, 2015.

[174] Steve Ranger. The undercover war on your internet secrets: How online surveillance cracked our trust in the web, 2016. `https://web.archive.org/web/20160612190952/http://www.techrepublic.com/article/the-undercover-war-on-your-internet-secrets-how-online-surveillance-cracked-our-trust-in-the-web/`.

[175] Michael G Reed, Paul F Syverson, and David M Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected areas in Communications*, 16(4):482–494, 1998.

[176] Panama Papers reporting team The Guardian. What are the Panama Papers? A guide to history's biggest data leak, April 2016.

[177] Ronald L Rivest and Butler Lampson. SDSI-a simple distributed security infrastructure. In *Crypto*, 1996.

[178] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[179] Paul Roberts. Phony SSL Certificates issued for Google, Yahoo, Skype, Others. `https://threatpost.com/phony-ssl-certificates-issued-google-yahoo-skype-others-032311/75061/`, 2011.

[180] Seth Rosenblatt. NSA likely targets anybody who's 'Tor-curious', July 2014. `https://www.cnet.com/news/nsa-likely-targets-anybody-whos-tor-curious/`.

[181] Volker Roth, Benjamin Güldenring, Eleanor Rieffel, Sven Dietrich, and Lars Ries. A secure submission system for online whistleblowing platforms. In *International Conference on Financial Cryptography and Data Security*, pages 354–361. Springer, 2013.

[182] Scott Ruoti, Jeff Andersen, Daniel Zappala, and Kent E. Seamons. Why Johnny Still, Still Can't Encrypt: Evaluating the Usability of a Modern PGP Client. *CoRR*, abs/1510.08555, 2015.

[183] Mark D Ryan. Enhanced certificate transparency and end-to-end encrypted mail. *Cryptology ePrint Archive*, 2013.

[184] M Angela Sasse and Ivan Flechais. Usable security: Why do we need it? How do we get it? O'Reilly, 2005.

[185] Quirin Scheitle, Oliver Gasser, Theodor Nolte, Johanna Amann, Lexi Brent, Georg Carle, Ralph Holz, Thomas C. Schmidt, and Matthias Wählisch. The Rise of Certificate Transparency and Its Implications on the Internet Ecosystem. In *Proceedings of the Internet Measurement Conference 2018*, IMC '18, page 343–349, New York, NY, USA, 2018. Association for Computing Machinery.

[186] Svenja Schröder, Markus Huber, David Wind, and Christoph Rottermanner. When Signal hits the fan: On the usability and security of state-of-the-art secure mobile messaging. In *European Workshop on Usable Security. IEEE*, 2016.

[187] James Scott, Jon Crowcroft, Pan Hui, and Christophe Diot. Haggle: A networking architecture designed around mobile users. In *Conference on Wireless On-demand Network Systems and Services (WONS)*, pages 78–86, 2006.

[188] SecureDrop. About SecureDrop. `https://securedrop.org/faq/about-securedrop/`.

[189] SecureDrop. Piloting SecureDrop Workstation for Qubes OS, February 2020. `https://securedrop.org/news/piloting-securedrop-workstation-qubes-os/`.

[190] Jérôme Segura. Digital certificates and malware: a dangerous mix, February 2013. `https://blog.malwarebytes.com/threat-analysis/2013/02/digital-certificates-and-malware-a-dangerous-mix/`.

[191] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. In *International Workshop on Information Hiding*, pages 36–52. Springer, 2002.

[192] Steve Sheng, Levi Broderick, Colleen Alison Koranda, and Jeremy J. Hyland. Why Johnny still can't encrypt: Evaluating the usability of email encryption software. In *Symposium On Usable Privacy and Security (SOUPS)*, pages 3–4, 2006.

[193] Signal. What is a safety number and why do I see that it changed?, 2021. https://support.signal.org/hc/en-us/articles/360007060632-What-is-a-safety-number-and-why-do-I-see-that-it-changed.

[194] David Sommer, Aritra Dhar, Luka Malisa, Esfandiar Mohammadi, Daniel Ronzani, and Srdjan Capkun. Deniable upload and download via passive participation. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 649–666, 2019.

[195] SophosLabs. "Who's your Verisign?" – Malware faking digital signatures, June 2010. https://nakedsecurity.sophos.com/2010/06/23/trojbhoqp-verisign/.

[196] Der Spiegel. Former US Official Reveals Risks Faced by Internal Critics. http://www.spiegel.de/international/world/ex-us-official-reveals-risks-faced-by-internal-govt-critics-a-1093360-2.html, 2016. https://perma.cc/8TKA-76BW.

[197] Data Encryption Standard et al. Data encryption standard. *Federal Information Processing Standards Publication*, 112, 1999.

[198] Emily Stark, Joe DeBlasio, and Devon O'Brien. Certificate transparency in Google Chrome: Past, present, and future. *IEEE Security & Privacy*, 19(6):112–118, 2021.

[199] Emily Stark, Ryan Sleevi, Rijad Muminovic, Devon O'Brien, Eran Messeri, Adrienne Porter Felt, Brendan McMillion, and Parisa Tabriz. Does certificate transparency break the web? Measuring adoption and error rate. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 211–226. IEEE, 2019.

[200] Emily M. Stark. Certificate transparency: a bird's-eye view, 2020.

[201] Ryan Stedman, Kayo Yoshida, and Ian Goldberg. A user study of Off-the-Record messaging. In *Proceedings of the 4th Symposium on Usable privacy and security (SOUPS)*, pages 95–104. ACM, 2008.

[202] The Sun. The sun launches whistleblowers' charter. https://www.thesun.co.uk/archives/news/142181/the-sun-launches-whistleblowers-charter/, 2015. https://perma.cc/8PVD-L7EW.

[203] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 526–545. IEEE, 2016.

[204] Süddeutsche Zeitung. So erreichen Sie das Investigativ-Team der Süddeutschen Zeitung, 2020. https://www.sueddeutsche.de/projekte/kontakt/.

[205] The NYT Open Team. To Serve Better Ads, We Built Our Own Data Program, 2020. https://open.nytimes.com/to-serve-better-ads-we-built-our-own-data-program-c5e039bf247b.

[206] Einar Thorsen. Surveillance of journalists/encryption issues. *The International Encyclopedia of Journalism Studies*, pages 1–7, 2019.

[207] New York Times. Russian Bank Reformer Dies After Shooting. https://www.nytimes.com/2006/09/15/world/europe/15russia.html?_r=1&oref=slogin.

[208] New York Times. Manning Sentenced to 35 Years for a Pivotal Leak of U.S. Files. https://www.nytimes.com/2013/08/22/us/manning-sentenced-for-leaking-government-secrets.html, 2013. https://perma.cc/8SEL-9KKL.

[209] New York Times. Got a confidential news tip?, 2018. https://www.nytimes.com/tips.

[210] Alin Tomescu and Srinivas Devadas. Catena: Efficient non-equivocation via bitcoin. In *Symposium on Security and Privacy (SP)*, pages 393–409. IEEE, 2017.

[211] Anton Troianovski and Valeriya Safronova. Russia takes censorship to new extremes, stifling war coverage, 2022. https://www.nytimes.com/2022/03/04/world/europe/russia-censorship-media-crackdown.html.

[212] Lokman Tsui and Francis Lee. How journalists understand the threats and opportunities of new technologies: A study of security mind-sets and its implications for press freedom. *Journalism*, 22(6):1317–1339, 2021.

[213] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. SoK: Secure messaging. In *IEEE Symposium on Security and Privacy (SP)*, pages 232–249, 2015.

[214] Nik Unger and Ian Goldberg. Deniable key exchanges for secure messaging. In *Proceedings of the 22nd ACM Sigsac conference on computer and communications security*, pages 1211–1223, 2015.

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

[216] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. SGAxe: How SGX fails in practice, 2020. `https://sgaxeattack.com/`.

[217] Diana A. Vasile, Martin Kleppmann, Daniel R. Thomas, and Alastair R. Beresford. Ghost trace on the wire? Using key evidence for informed decisions. In *27th International Workshop on Security Protocols (SPW)*, volume LNCS 12287. Springer, 2019.

[218] Elham Vaziripour, Devon Howard, Jake Tyler, Mark O'Neill, Justin Wu, Kent Seamons, and Daniel Zappala. I Don't Even Have to Bother Them! Using Social Media to Automate the Authentication Ceremony in Secure Messaging. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[219] Elham Vaziripour, Justin Wu, Mark O'Neill, Jordan Whitehead, Scott Heidbrink, Kent Seamons, and Daniel Zappala. Is that you, Alice? A usability study of the authentication ceremony of secure messaging applications. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 29–47, 2017.

[220] Igor Vobic and Melita Poler Kovacic. Watchdog journalism and confidential sources: A study of journalists' negotiation of confidentiality with their sources. *Teorija in Praksa*, 52(4):591–611, 2015.

[221] Daniel T. Wagner, Andrew Rice, and Alastair R. Beresford. Device Analyzer: Understanding smartphone usage. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*, pages 195–208. Springer, 2013.

[222] Mei Wang, Kun He, Jing Chen, Zengpeng Li, Wei Zhao, and Ruiying Du. Biometrics-authenticated key exchange for secure messaging. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2618–2631, 2021.

[223] Peng Wang, Carl Hua, and Michael Zochowski. Benchmarking hash and signature algorithms, 2019. `https://medium.com/logos-network/benchmarking-hash-and-signature-algorithms-6079735ce05`.

[224] Stephenson Waters. The effects of mass surveillance on journalists' relations with confidential sources: A constant comparative study. *Digital Journalism*, 6(10):1294–1313, 2018.

[225] WhatsApp. Security code change notification, 2021. `https://faq.whatsapp.com/general/security-and-privacy/security-code-change-notification/?lang=en`.

[226] WhatsApp Inc. Connecting One Billion Users Every Day, July 2017. `https://blog.whatsapp.com/10000631/Connecting-One-Billion-Users-Every-Day`.

[227] Alma Whitten and J. D. Tygar. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. In *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, SSYM'99, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.

[228] Alma Whitten and J. Doug Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *USENIX Security Symposium*, volume 348, 1999.

[229] Wikileaks. Submit documents to WikiLeaks, 2016. `https://wikileaks.org/Press.html#submit_help_contact`.

[230] Wikipedia. Indictment and arrest of Julian Assange, 2019. `https://en.wikipedia.org/wiki/Indictment_and_arrest_of_Julian_Assange`.

[231] WIRED. How to tip WIRED anonymously, 2019. `https://www.wired.com/securedrop/`.

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

[233] Fengli Xu, Zhen Tu, Yong Li, Pengyu Zhang, Xiaoming Fu, and Depeng Jin. Trajectory recovery from ash: User privacy is not preserved in aggregated mobility data. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 1241–1250, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

[234] Philip Zimmermann. PGP (tm) User's Guide. *Massachusetts Institute of Technology*, 2(2), 1994.

[235] Philip R Zimmermann. *The official PGP user's guide*. MIT press, 1995.