Rethinking Information Privacy for the Web

by

Matthew Ryan Tierney

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September, 2013

Professor Lakshminarayanan Subramanian

© MATTHEW RYAN TIERNEY
SOME RIGHTS RESERVED, 2013 (SEE APPENDIX 6.2.3).

Dedication

TO CHRISTINA

AND

MY PARENTS

Acknowledgments

FIRST AND FOREMOST, I would like to thank my advisor, Lakshmi Subramanian. His willingness to adapt to and encourage my ever-evolving interests enabled the body of this thesis to come to fruition.

I would also like to thank Jinyang Li and Helen Nissenbaum, who have been my co-advisors at NYU. Jinyang's diversified set of deep knowledge enabled her to provide unparalleled guidance as I explored a variety of potential research interests.

Helen, of course, is entirely responsible for re-wiring the way I think about privacy. Her work on contextual integrity permeates this research and continues to affect how I operate in everyday life. To paraphrase and apply a farewell address by Dietrich Bonhoeffer, "that you were my teacher for many sessions is a thing of the past; but that I may call myself your pupil remains still."

The members of the NYU Privacy Research Group provided a second academic home during these years. Without that group – and Helen's gift for academic matchmaking – I would not have had the privilege to work with Luke Stark on Lockbox or Ian Spiro on Cryptagram.

The cohort on the 7th Floor of 715/719 Broadway formed my primary academic home at NYU. The members of the Networking and Wide-Area Systems Group

helped me to grow tremendously: Sunandan Chakraborty, Jay Chen, Aditya Dhananjay, Eric Hielscher, Paul Gazzillo, David Iserovich, Vipin Jain, Shankar Kalyanaraman, Arthur Meachem, Christopher Mitchell, Michael Paik, Russell Power, Nektarios Paisios, Alex Rubinsteyn, Ashlesh Sharma, Robert Soule, Yair Sovran, and Nguyen Tran. Robert Grimm and Allan Gottlieb always provided the most interesting interrupts. And Leslie Cerve, who always brought a breath of fresh air to our floor.

And the professors who inspired me to pursue graduate school in the first place, supporting me tremendously as an undergraduate: Matt Welsh, Margo Seltzer, and Jim Waldo. I am so thankful that in the years since I graduated, Matt and Jim have been advisors in both work and life.

I am thankful to Rupa Krishnan for hosting me as an intern twice at Google. Rupa introduced me to a number of individuals and interesting projects, applying the engineering salve that I had been looking for all along. I am also thankful to Googlers Adam Bender, Marc Donner, Anand Kanagala, and B. Narendran, who always worked hard to enable me grow as an engineer and researcher.

My family provided much needed encouragement. Bruce and Unae became parents to me during these years. They have demonstrated empathy and been thoughtful listerners and supporters of my journey.

And Richard and Dorothy, who have prioritized my education and happiness always, time and age continue to affirm how wonderful you have been to me since the beginning.

My wife, Christina, has always sought to understand and support me through the course of this thesis. While she had her own uphill battles in her career, she always

made time to ensure I was as healthy and happy as could be. I can think of no better partner with whom I could have shared this difficult journey. I have reached the finish line because of her constant source and reflection of love.

Abstract

Hanni M. Fakhoury, staff attorney with the Electronic Frontier Foundation, has argued against Supreme Court Justice Samuel Alito's opinion that society should accept a decline in personal privacy with modern technology, "Technology doesn't involve an 'inevitable' tradeoff [of increased convenience] with privacy. The only inevitability must be the demand that privacy be a value built into our technology" [40]. Our position resonates with Mr. Fakhoury's assertion for rethinking information privacy for the web. In this thesis, we present three artifacts that address the balance between usability, efficiency, and privacy as we rethink information privacy for the web.

In the first part of this thesis, we propose the design, implementation and evaluation of Cryptagram, a system designed to enhance online photo privacy. Cryptagram enables users to convert photos into encrypted images, which the users upload to Online Social Networks (OSNs). Users directly manage access control to those photos via shared keys that are independent of OSNs or other third parties. OSNs apply standard image transformations (JPEG compression) to all uploaded images

so Cryptagram provides image encoding and encryption protocols that are tolerant to these transformations. Cryptagram guarantees that the recipient with the right credentials can completely retrieve the original image from the transformed version of the uploaded encrypted image while the OSN cannot infer the original image. Cryptagram's browser extension integrates seamlessly with preexisting OSNs, including Facebook and Google+, and currently has over 400 active users.

In the second part of this thesis, we introduce the design of Lockbox, a system designed to provide end-to-end private file-sharing with the convenience of Google Drive or Dropbox. Lockbox uniquely combines two important design points: (1) a federated system for detecting and recovering from server equivocation and (2) a hybrid cryptosystem over delta encoded data to balance storage and bandwidth costs with efficiency for syncing end-user data. To facilitate appropriate use of public keys in the hybrid cryptosystem, we integrate a service that we call KeyNet, which is a web service designed to leverage existing authentication media (e.g., OAuth, verified email addresses) to improve the usability of public key cryptography.

In the third part of this thesis, we propose a new system, Compass, which realizes the philosophical privacy framework of *contextual integrity (CI)* as a full OSN design. CI), which we believe better captures users privacy expectations in OSNs. In Compass, three properties hold: (a) users are associated with *roles* in specific *contexts*; (b) every piece of information posted by a user is associated with a specific *context*; (c) *norms* defined on roles and attributes of posts in a context govern how information is shared across users within that context. Given the definition of a context and its corresponding norm set, we describe the design of a compiler that converts the human-readable norm definitions to generate appropriate information

flow verification logic including: (a) a compact binary decision diagram for the norm set; and (b) access control code that evaluates how a new post to a context will flow. We have implemented a prototype that shows how the philosophical framework of contextual integrity can be realized in practice to achieve strong privacy guarantees with limited additional verification overhead.

Contents

Listing of figures

Listing of tables

Listing of appendices

1

Introduction

In his concurrence to the United States v. Jones GPS decision, Supreme Court Justice Samuel Alito wrote that "New technology may provide increased convenience or security at the expense of privacy, and many people may find the tradeoff worthwhile. And even if the public does not welcome the diminution of privacy that new technology entails, they may eventually reconcile themselves to this development as inevitable" [6]. But Hanni M. Fakhoury, staff attorney with the Electronic Frontier Foundation, responds, "Technology doesn't involve an 'inevitable' tradeoff with privacy. The only inevitability must be the demand that privacy be a value built into our technology" [40]. For our thesis, we aim to demonstrate the strength Fakhoury's assertion in the modern web.

In this thesis, we present three artifacts that address the balance between convenience, utility and privacy when considering photo sharing, file sharing, and the fundamental architecture of online social networks. We demonstrate the feasiblity of balancing these three values of convenience, utility and privacy when designing systems for the web.

1.1 Photo Sharing

While Online Social Networks (OSNs) enable users to share photos easily, they also expose users to several privacy threats from both the OSNs and external entities. The current privacy controls on OSNs are far from adequate, resulting in inappropriate flows of information when users fail to understand their privacy settings or OSNs fail to implement policies correctly. OSNs may further complicate privacy expectations when they reserve the right to analyze uploaded photos using automated face identification techniques.

In Chapter 2, we propose the design, implementation and evaluation of Cryptagram, a system designed to enhance online photo privacy. Cryptagram enables users to convert photos into encrypted images, which the users upload to OSNs. Users directly manage access control to those photos via shared keys that are independent of OSNs or other third parties. OSNs apply standard image transformations (JPEG compression) to all uploaded images so Cryptagram provides an image encoding and encryption mechanism that is tolerant to these transformations. Cryptagram guarantees that the recipient with the right credentials can completely retrieve the original image from the transformed version of the uploaded encrypted image while the OSN cannot infer the original image. The Cryptagram system uses a browser

extension to integrate seamlessly with preexisting OSNs, including Facebook and Google+, and has been installed by over 400 users.

1.2 File Sharing

Petabytes of data are replicated and shared through popular cloud storage service providers such as Dropbox and Google Drive. Nevertheless, end-to-end user privacy has remained an afterthought in popular services. Chapter 3 introduces the design of Lockbox, a system designed to provide end-to-end private file-sharing with the convenience of Google Drive or Dropbox. Lockbox minimizes cost and impact on the storage provider while maintaining file versioning and privacy through a transparent hybrid cryptosystem and a delta-to-object storage protocol, which stores encrypted delta-encoded data at the storage provider while maintaining local object blobs at the end-user devices. To facilitate appropriate use of private keys in the hybrid cryptosystem, we integrate a service that we call KeyNet, which is a web service designed to leverage existing authentication media (e.g., OAuth, verified email addresses) to improve the usability of public key cryptography.

1.3 Online Social Network Design

Privacy violations in online social networks (OSNs) have become more the norm than the exception. In Chapter 4, we argue that the conventional models of privacy in OSNs are fundamentally flawed and offer no specific privacy guarantees. We propose a completely new model of private information sharing using a refined abstraction of *circles* that embodies the philosophy of *contextual integrity* (CI). We contend

that the CI framework provides a cleaner match between user privacy and sharing intentions and also enables a verification framework to formally check correctness. We present the design of a social network that leverages the CI framework.

1.4 THESIS CONTRIBUTIONS

This thesis makes the following contributions:

1. We define q, p-Recoverability and present **a systematic exploration of the design space of protocols for embedding data into the spatial domain of images** through the lens of q, p-Recoverability.

2. We present the design and evaluation of **Cryptagram, a usable photo privacy solution for sharing private data through online social media**. Our preliminary users study includes over 400 active account installation of our Chrome extension.

3. We describe the design and implementation of Lockbox, **a system for lightweight private file-sharing across federated services**. Lockbox demonstrates how to reduce cloud storage requirements while retaining end-to-end file sharing privacy.

4. We propose the design of and evaluate an implementation of an **online social network privacy system that embodies contextual integrity**. We demonstrate how to close the gap between the philosophical framework and reality with the Compass norm compiler.

In short, this thesis provides systems that empower end-users to rethink how they manage their privacy on the web. It shows how to resolve the rumored tension between convenience, efficiency and privacy in the domains of photo sharing in online social networks, cloud file sharing, and online social network design. This work helps to support and encourage future innovation in web privacy.

2

Photo Privacy for Online Social Media

Petabytes of imagery data have been posted by users to Online Social Networks (OSNs) with Facebook alone receiving over 250 million photo uploads per day [99], storing 10,000 times more photos than the Library of Congress [53]. Users feel the need internally and externally (peer pressure) to share photos on OSNs given the convenience of usage and their immense popularity [19, 76, 101, 104, 120]. Users share personal and potentially sensitive photos on OSNs, thereby exposing users to a wide range of privacy threats from external entities and the OSN itself [34, 112, 118]. We consider two basic factors that trigger privacy concerns for end-users in OSNs.

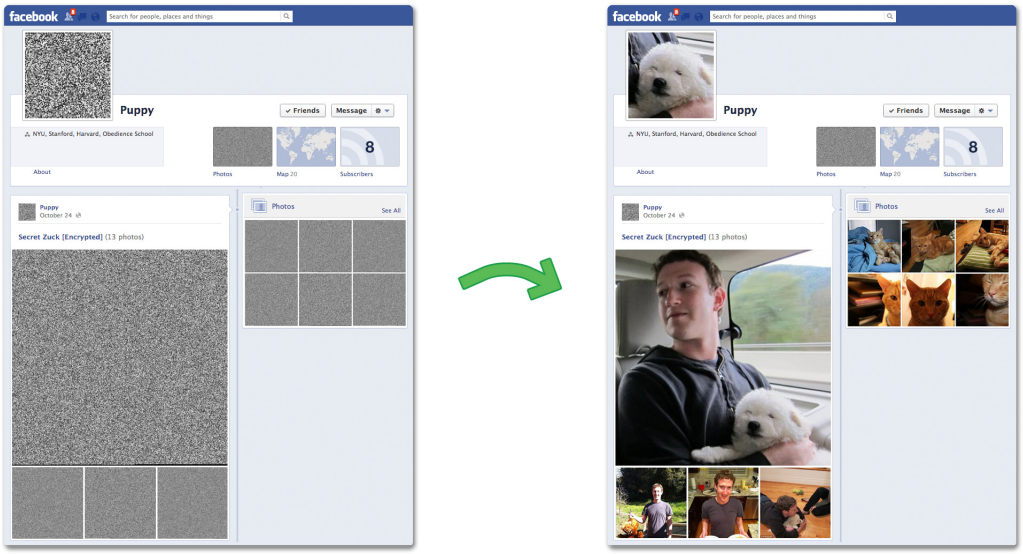


Figure 2.0.1: Example Cryptagram user experience. On the left, we show a social network with embedded Cryptagrams, uploaded by a user. A browser extension decrypts the images in place as shown on the right.

USER/SYSTEM ERRORS: A user who uploads an image to an OSN may wish to share it with only a select group of people, which OSNs partially satisfy with privacy settings. Contextual integrity [94] would state that the user is attempting to implement her own notion of appropriate information flows. But a recent study confirmed that Facebook users' impression of their sharing patterns and their true privacy settings are often inconsistent [64]. Moreover, an OSN may fail to correctly enforce their privacy settings, such as the case when Facebook exposed its own CEO's private photos in a systemwide glitch [34].

FACE IDENTIFICATION: A passive observer or a hosting OSN could extract large volumes of online photo uploads, indexing and discovering images within a corpus that belong to a specific user [118]. Mining of photo corpora can lead to the

unexpected disclosure of individuals' locations or their participation in events. Facial data mining incidents have resulted in litigation against OSNs and further weakened the integrity of the relationship between the social network and the individual [112].

In this chapter, we present the design, implementation and evaluation of Cryptagram, a system designed to address these photo privacy concerns in OSNs. A basic design goal of Cryptagram is to build a usable solution that can offer strong privacy guarantees for end-users that remains backwards compatible with existing OSN user interface designs. To maintain the conventional feel of an OSN, Cryptagram uses the abstraction of an image interface (RGBA pixels) to manipulate the core image formats used by OSNs. Cryptagram leverages an end-to-end encryption system to transport images, which are uploaded to OSNs. Figure 2.0.1 illustrates a specific example of how Cryptagram represents normal images as encrypted images.[1]

A challenge in the design of such an end-to-end image encryption/decryption mechanism is to be resilient to image transformations by the OSN. For instance Facebook converts all uploaded photos, regardless of original format, to JPEG, choosing quality settings without user input. The recipient of a Cryptagram image must be able to retrieve the original image from the OSN-transformed version. In this paper, we describe the notion of q, p-Recoverability (Section 2.2.1) which formalizes the aforementioned property that enables the assessment of embedding protocol designs. We describe a class of JPEG embedding protocols that can achieve the q, p-Recoverability property for different JPEG quality transformation levels. The

[1] In this example, a user has uploaded a single Cryptagram image per image in this Figure. OSNs typically recompress and resize images within their backend infrastructure, presenting the most bandwidth-friendly version (thumbnails) as they deem appropriate. In order to render the decrypted Cryptagrams for thumbnails, Cryptagram infers from URL of the thumbnail how to fetch the full-size image, which Cryptagram fetches and decompresses when a user indicates our extension should do so.

top-down techniques that we discuss for designing q, p-Recoverable protocols can also be applied to lossless image compression formats.

Cryptagram addresses a problem that is fundamentally different from conventional image steganography. While steganography aims to hide data in plainsight and *avoid detection* [48], Cryptagram makes obvious that it is hiding data with the added aim of efficiently transporting bits in the image medium while being robust to image transformations. Despite the differences in problem definition, steganography does have the same mechanical use as Cryptagram for transporting bits in an image. When comparing the effective efficiency of our approach to steganography, Cryptagram packs many more bits per pixel (Section 2.5).

Cryptagram differs significantly from the recent work on photo privacy, P3 [106]. Unlike P3, Cryptagram operates completely in encrypted bit space and does not reveal sensitive cleartext data of photos to external entities (Section 5.1). Cryptagram also does not rely on third-party providers for providing photo privacy.

We present several key results in our evaluation. For JPEG Cryptagram images uploaded to Facebook, we find that JPEG compression quality for those high entropy images is in the range of 76 to 86 (for natural images, usually quality is 74). Given these recompression target ranges, we demonstrate JPEG embedding protocols that, in tandem with error-correcting codes, achieve an effective efficiency of 3.06 bits per pixel, which is 2.68$\times$ greater than the best related work. We also summarize a study of recoverability when recompressng already compressed images. We further illustrate a design point comparison of recoverability versus filesize expansion when comparing JPEG and webp lossy compression formats.

Our end-to-end Cryptagram system has been deployed to the web. Figure 2.0.2

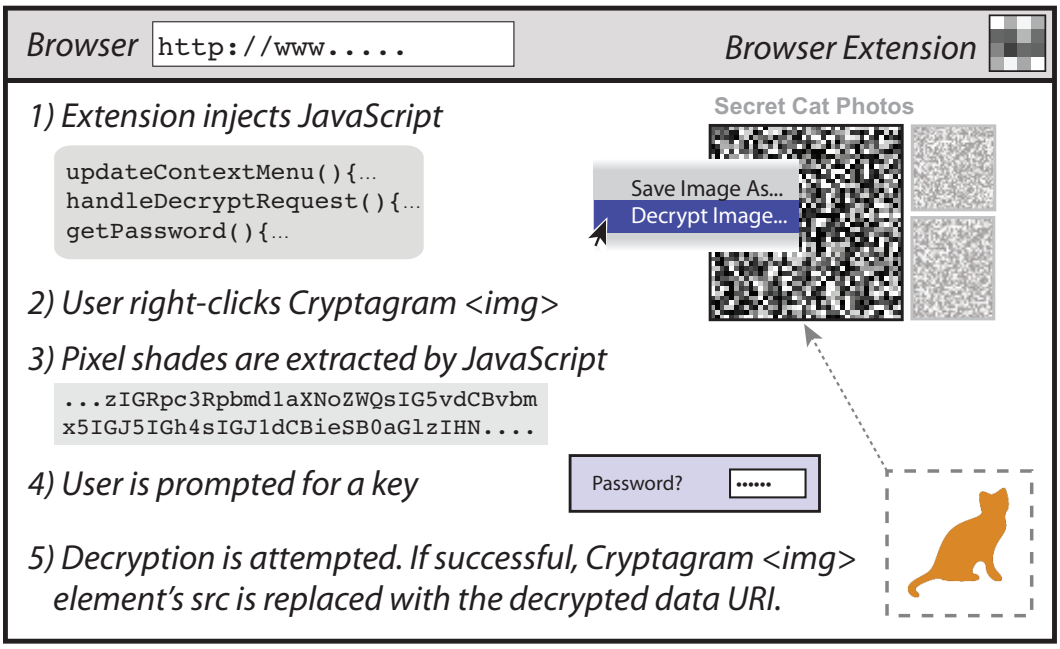


Figure 2.0.2: An overview of the Cryptagram user experience.

summarizes a user's experience with the current decoder. Our decoder browser extensions integrate seamlessly with existing OSNs including Facebook and Google+ while being compatible with their complicated DOM structures. We have nearly 400 active users of our decoder extension and over 300 users have agreed to our IRB-approved study through which they submit high-level data about their on-going image encrypting and decrypting habits with Cryptagram.

2.1 System Model

2.1.1 Problem Statement

The basic problem that Cryptagram aims to address can be stated as follows: Two users U and V are members in an OSN and have a shared key k (e.g., password),

independent of the OSN. U wants to use the OSN to share an image I with V but does not intend to upload I to the OSN since the OSN or other unauthorized users may also be able to view I. Instead, U needs an encryption mechanism that can transform I into an encrypted image I', which V can retrieve, decrypt and obtain I (using the shared key k). The key challenge is that when U uploads an encrypted image I', the OSN can apply image transformations and the image V downloads may significantly differ from I'. Hence, the sharing mechanism needs to be resilient to image transformations.

To better understand the transformation-resilient image encryption problem, we outline the basic image encoding and decoding steps used by Cryptagram and the property that Cryptagram aims to achieve:

- A user U encrypts a to-be-shared cleartext image, I, using a strong block cipher with a secret key, k, to produce a byte sequence, $E(I, k)$. This k may be human-readable (a password) or part of a hybrid cryptosystem in which the k is generated and shared using public key cryptography.

- An image encoding protocol, C, embeds $E(I, k)$ into the spatial domain of an image, $I_m = C(E(I, k))$ which the OSN transforms as $T(I_m)$. In this chapter, we restrict T to be the identity transformation (lossless format) or a standard lossy image compression. We use JPEG for the lossy format in much of the evaluation since that is a commonly used standard across OSNs.

- An OSN user V who is an authorized recipient of the image needs to be aware of the block cipher secret key k. Using this key k and an image decoding algorithm, V should be able to successfully decode I from a transformed version

$T(I_m)$. Here, we aim to achieve *recoverability* (intuitively, data integrity of the embedded message), which in the case of a lossless format is tautologically true sans other transformations. For JPEG, we aim for q, p-Recoverability property: given a minimum quality level q of the transformed image $T(I_m)$, the decoding protocol should enable V to decode the original image I with high probability, p. We denote this recoverability probability using p, where the ideal case is when $p = 1$; however, achieving $p = 1$ may not always be feasible.

- Adversary, Eve, who passively observes $T(I_m)$ should not learn anything about I.

2.1.2 Design Goals

The aforementioned problem statement highlights two key design goals of Cryptagram: ***data confidentiality*** and ***probabilistic data integrity for lossy images***. For data confidentiality, Cryptagram leverages trusted algorithms that users may use to ensure that data has been encoded with a strong proofs of security. For probabilistic data integrity, since Cryptagram aims to create images for use on OSNs, we can relax the constraint of traditional information security data integrity [44] for lossy image formats such as JPEG. This relaxation enables the Cryptagram design to incorporate features that demonstrate a spectrum of data integrity when a social network transforms uploaded images.

But given these goals, should U and V use an OSN to share personal images at all? We accept as a constraint that many users desire the convenience of social networks [76]. This "convenience" constraint raises the following design goals that

Cryptagram meets:

- **Usable:** We aim to offer a system that affords users intuitive privacy on top of the images that they share on OSNs. While several offline approaches exist to preserve privacy (e.g., PGP [135]), Cryptagram makes it possible for users to create and share private photos without disrupting the OSN experience.
- **Widely Deployable and Applicable:** To gain wide adoption, we have created a cross-browser, cross-platform, cross-image format system that enables Cryptagram to be used as both an encoder and decoder. The reduced friction to creating and accessing Cryptagram images removes the barrier to broader use of the technology.
- **Efficient:** Compared to alternative methods, we present a system that offers significantly more data storage for a given file size or image dimensions.

2.1.3 Security Overview

Threat #1: Facial Data Mining

In this threat, the adversary is the OSN, whose aim is to execute facial data mining algorithms on uploaded images. In recent years, as social networks' corpi of images have grown dramatically, this is a serious concern for the privacy-conscious individual.

Approach. We have devised a scheme that reveals no information about the original image to the OSN. As we discuss in Section 2.3, the use of our embedding

algorithm by nature thwarts facial data mining by embedding the cleartext (e.g., facial) data indirectly as encrypted bits in the spatial domain of the transport image.

SECURITY GUARANTEES. With the use of a block cipher to transform the secret message, Cryptagram retains the strength of the underlying security properties of the chosen algorithm. With the use of public key cryptography users can retain cryptographic strength while leveraging a trusted, separate channel to bootstrap their sharing.

THREAT #2: MISCONFIGURED PRIVACY CONTROLS

OSNs may fail to correctly deploy access control policies or users may accidentally misconfigure confusing access controls. The use of Cryptagram creates a separate channel of communication to ensure, with cryptographic strength, that only intended recipients see the cleartext photo. With the correct use of Cryptagram, an OSN could suffer a full system breach and encrypted images would remain private.

LIMITATIONS

DETECTING AND BLOCKING CRYPTAGRAM IMAGES. Cryptagram does not address the problem of an OSN detecting and inhibiting the upload of all Cryptagram images. Steganography may be proposed in this scenario but problem redefinition and the tradeoff in efficiency make steganography an inappropriate application.

UNSUPPORTED TRANSFORMATIONS. Though Cryptagram images are robust to varying degrees of JPEG compression, they does not support many other

transformations. For example, cropping or rescaling a Cryptagram will generally break its encoding.

BRUTE-FORCE CRYPTOGRAPHIC ATTACK. Cryptagram users who choose weak passwords in the symmetric key scheme can be attacked with dictionary or brute-force techniques. To address this limitation, we encourage users to abide by strong password creation techniques [87, 105] when using symmetric key credentials. When using public key cryptography, we encourage users to leverage the use of a public key infrastructure that is coupled with Cryptagram as we follow Key Continuity Management practices [51].

COPY AND PASTE. Users who gain access to cleartext images can copy and paste those images to whomever they choose. We believe this problem will persist despite any attempts, short of controlling all hardware at the disposal to humans accessing social networks.

2.2 IMAGE FORMATS IN OSNS

Several image formats are used across OSNs. While Facebook uses only the JPEG format to store images (and, moreover, strips uploaded images of EXIF data), Google+ and other networks allow for a variety of lossless (e.g., PNG) and lossy (e.g., webp) formats. Our goal is to design a generic photo privacy solution that can work across different image formats. While lossless compression techniques are relatively easier to handle, determining an image encryption/decryption mechanism in the face of a lossy transformation is much more challenging. Given the popularity and broad

use of JPEG, we use JPEG as our primary image format to describe the design of Cryptagram. We show how Cryptagram can be easily applied for other image formats including lossy image formats like webp.

Our design primarily focuses on embedding data in the spatial dimensions of an image. We define an embedding protocol to be an algorithm that describes the creation of a sequence of bits and how those bits are embedded into the spatial domain (pixels) of an image. We design embedding algorithms that work in a top-down fashion; that is, the data to be embedded is written into the spatial domain of an image on a pixel level rather than in any protocol-specific manner. We believe that a top-down approach allows us to meet the aim for wide deployablility and applicability in terms of implementation, testing and future image formats. The top-down API means that the design of codecs can apply or be tested across multiple formats with ease. When codec design depends on DCT coefficients, for instance, there are non-intuitive programming interfaces that would be required to make that facility addressable to the PNG format and not just JPEG, webp, and other DCT-coefficient based compression algorithms.

Assuming a passive adversary, this approach is a valid solution to the security threats that we outlined in the previous section. This is an especially prudent design choice considering that lossy image transformations will most intuitively aim to preserve higher order features of an image rather than its bit-wise representation.

The generic interface to the image is thus the four possible color channels red, green, blue, and alpha as well as their corresponding bit value. For JPEG, this means up to eight bits per the first three color channels. For PNG, we have up to 16 bits per channel for all four possible channels.

2.2.1 DEFINING q, p-RECOVERABILITY FOR JPEG

JPEG image transformations are inherently lossy in nature. With the aim of probabilistic data integrity, we make concrete the goal of relaxing the constraints of traditional notions of information security data integrity [44] for embedding data in JPEG.

We define the q, p-*JPEG Recoverability* (or, simply q, p-*Recoverability*) property of embedding protocols as follows: given a minimum quality level q that an OSN preserves in a transformation T of an uploaded image, an authorized recipient should be able to decode the original image with high probability p, where in the ideal case $p = 1$. The concept of q, p-Recoverability can also be applied to other lossy image transformations though the corresponding attainable values of q and p are dependent on transformation T.

In the context of JPEG images, we define a Cryptagram protocol as a message-JPEG encoder G and JPEG-message decoder G'. Given an input image[2] I, the first step in Cryptagram encoding is to convert the image into an encrypted sequence of bits $m = E(I, k)$, for clear-text image I and a block cipher key k. We refer to the input to the JPEG encoder as a sequence of bits m. Given m, the protocol encodes m in the spatial domain of a JPEG image, $I_m = G(m)$. JPEG (denoted by the function T, its inverse for decompression is T') compresses I_m at quality q to produce a sequence of bits, $T(I_m, q)$.

The recipient uses a two step decoding mechanism to retrieve an encrypted set of bits m': (a) the first step involves using the decompression step T' to produce

[2]We mean that I is a sequence of bits that represent an image format that a browser can render. Notably, Cryptagram's embedding can be a used with any arbitrary message I for delivering a message via the spatial domain of a transport image.

$$I_m = G(m)$$
$$G'(T'(T(I_m, q))) = m' =_p m \implies$$
$$G \text{ is } q, p\text{-Recoverable}$$

Figure 2.2.1: q, p-Recoverability in a nutshell.

$T'(T(I_m, q))$; (b) the second step involves using the JPEG-message decoder G' to retrieve an encrypted sequence of bits $m' = G'(T'(T(I_m, q)))$. Ideally, m' should match m; if they do, the recipient can use the secret key k to decrypt m' to retrieve the original input message. However given the lossy nature of the transformations, the message-JPEG encoding and JPEG-message decoding steps may not always succeed. Here, we use the term p to denote the probability that the algorithm successfully decodes the input bit sequence m. Mathematically, we denote this as: $m' =_p m$. If this constraint holds, then we define the protocol to be q, p-Recoverable. By considering a large sample set of input messages, we can statistically estimate the value of p for a given quality threshold q. The aim of Cryptagram is to identify q, p-Recoverable protocols that attain p close to one for low quality values and a high bits per pixel ratio. We summarize these ideas in Figure 2.2.1.

2.3 System Design

2.3.1 Lossy Images

To discuss how to embed data into a lossy image, we focus on the JPEG compression algorithm, though our design principles apply to other lossy formats.

How should one embed bits into the spatial domain of an image? To approach

this challenge, we develop a mapping of bits to colors for specific pixels in an image. Intuitively, when choosing points (coordinates) in the color space to represent bits, we leverage the observation that the lossy codec may *shift* an initially embedded point (pixel's color) during encoding and decoding an image; however, the sphere in the color space within which that point may move does not overlap with other point-centered spheres. This is to say that when choosing what values to embed and how to coordinate pixel values, protocol designers must be sensitive to the assumption that the lossy codec will shift values within spheres in a color space. This intuition guides our JPEG design discussion below but, more importantly, is the generally applicable principle for Cryptagram protocol design.

The principal unit of embedding in Cryptagram is the Cryptagram pixel block (CPB). Multiple CPBs must fill or pack a 8×8 JPEG pixel block (JPB) for each channel of JPEG (luminance, chrominance red and chrominance blue), which is the "atomic unit" of pixels that undergoes JPEG compression [125]. We consider how to pack bits into the spatial domain of JPEG given two goals: *(1) efficient bit packing (increasing the number of bits per pixel)* and *(2)* q, p*-Recoverability*.

Embedding in the Spatial Domain

Cryptagram Pixel Blocks For protocol design we examine how to manipulate 64 pixels to embed bits efficiently. We embed symbols into the 64-pixel JPBs for each YC_bC_r channel with multiple Cryptagram pixel blocks (CPBs) per JPB. A CPB could be any shape that packs into a JPB. For our discussion, we consider 1×1 and 2×2 CPBs.

The composition of a CPB thus is a shape description, $w \times h$ (width, height) and

a set of rules, R, for translating a symbol, x, or set of bits ($x = b_0, \ldots, b_{|x|}$) into red, green, and blue tuples (r, g, b) that we embed in each pixel of the CPB according to the appropriate $RGB \rightarrow YC_bC_r$ conversion. For simplicity, we represent the CPB embeddings for each channel as $L^{R_L}_{w \times h}$, where R_L are the rules that correspond to the channel, L, how to embed x to color values for L.

Because JPEG compression applies different downsampling and quantization matrices to luminance and chrominance channels (but applies the same compression to the two chrominance channels), we express the embedding protocol for a CPB as:

$$\left(Y^{R_Y}_{w_Y \times h_Y}, C^{R_C}_{w_C \times h_C}\right)$$

where Y corresponds to luminance and C to chrominance channels.

The rule set, R provides a large space for Cryptagram protocol designers. Intuitively, the composition of rules becomes a choice of three parameters: (1) how many bits to embed, (2) the number of discretizations to use (for which the number of bits to embed determines the lower-bound) in the color space, and (3) the choice of which discretization values from the color space to use. In short, we determine how many colors to use, what values they represent, and the resulting bitrate.

The JPEG compression algorithm compresses least the luminance channel of the three yielded by the $RGB \rightarrow Y'C_bC_r$ transformation. If we choose only to discretize values in luminance, we have an effective range of $[0, 255]$, which corresponds to "grayscale". We denote this scenario as $\left(Y^{R_Y}_{w_Y \times h_Y}, C^0\right)$, using C^0 to denote that chrominance is not used.

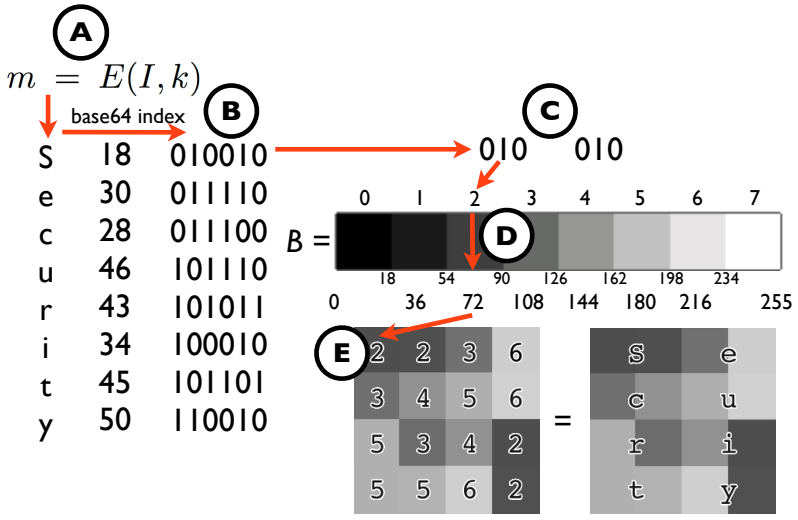


Figure 2.3.1: Encoding algorithm illustration. We demonstrate how Cryptagram maps an encrypted message's sequence of bits (Steps A-C) to color values (Step D) and how those correspond to embedded pixels (Step E).

ON CHROMINANCE EMBEDDING. When considering the use of the chrominance channels in the embedding protocol, there are several complications to address in this proposal. As described in the JPEG specification, the two chrominance channels are stored with significantly less fidelity than luminance. Both chrominance channels are down-sampled (by default in `libjpeg`, 2:1) and a more aggressive quantization table is used to further reduce the number of bits that need to be stored [125]. Intuitively, the chrominance channels are less efficient for embedding data in the spatial domain.

ENCODING ALGORITHM We demonstrate the embedding algorithm in Figure 2.3.1. As we discuss the embedding algorithm at a high-level, we will refer to the concrete demonstration in that figure.

The first step of the encoding algorithm transforms the input clear-text image I into a sequence of encrypted bits m using a shared key k such that $m = E(I, k)$. Here, we use a standard block cipher algorithm AES in CCM mode (128 bit keys and 64 bit tag size). The encoding algorithm from this point chooses a small collection of bits at a time and converts these bits into Cryptagram pixel blocks. Figure 2.3.1 Step A shows how our example encrypted output message m is the sequence of characters, "Security." Using the base64 representation of the character, we know that the sequence of bits for each character is shown under Step B. We then show in Step C how the sequence of bits for the first character (S's representation as 010010) can be split into two three-bit sequences, the aforementioned "small collection of bits." Using a chunked grayscale color spectrum, we map the three-bits to an index in the array of values. The index's group representative (in this case at Step D, it's the grayscale value of 72) is what is embedded for the appropriate pixels, as shown in Step E. In this example, we continue to pluck off three bits at a time, for Steps B and C, then map those three bits values to grayscale values in Step D. Finally, we continue to embed the values left-to-right, top-to-bottom in this simple example for Step E. We have used a 2×2 CPB for this illustration, which packs perfectly into a standard 8×8 JPB. An alternative format could have used 1×1 CPBs, shading one pixel instead of four in Step E.

Figure 2.3.2 illustrates at a high-level where data is embedded in a typical Cryptagram image format.

We make the above example more concrete in the following formalism. An embedding algorithm, a, assumes that b bits will be embedded per CPB. Given b, a has a bijective mapping $B_L : \{0, 1, \ldots, 2^b\} \rightarrow L_a$ where $L_a \subseteq [0, 255]$. Given a

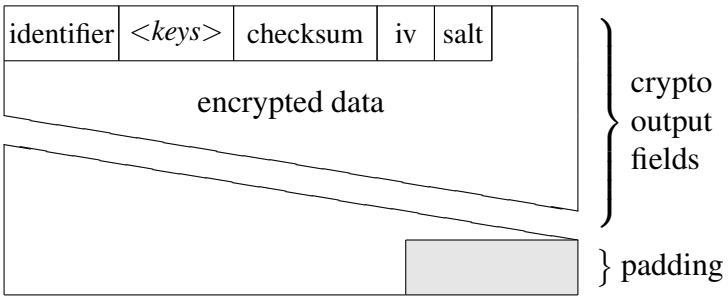


Figure 2.3.2: Layout of a Cryptagram. Each box is a sequence of shaded pixels representing the range of values for a particular protocol.

bit sequence, s, a uses B_L to map b-length substrings of s to the corresponding L channel values that will be embedded at that particular pixel. Given the notation we have introduced, B is a more specific case of the notion of rule sets R_L we presented earlier. B_L mappings underpin the designs we present in this chapter.

We can measure the efficiency of B_L based on b and the size of the channel's CPB to which the output of B_L mapped as $\frac{b}{|\text{CPB}|}$ bits per pixel, where $|\text{CPB}|$ is the number of pixels occupied by the CPB: $|\text{CPB}| = w \times h$.

From our discussion of the embedding protocol and the notion of q, p-Recoverability, we have laid the groundwork for how the designer's choice of protocol parameters ($d_{w,h}$, B, etc.) adjust the effective efficiency of the end-to-end protocol.

EXAMPLE ENCODINGS FOR REASONING ABOUT q, p-RECOVERABILITY. To demonstrate the tradeoff between efficiency (number of discretizations in B per CPB size) and q, p-Recoverability that we must consider in protocol design, we present two examples. The first example uses a $\left(Y^B_{1\times1}, C^0\right)$ CPB. As we translate

Name	Notation	Luminance-only B Mapping
Bin	$\left(Y^1_{1\times1}, C^0\right)$	$B : \{0, 1\} \rightarrow \{0, 255\}$
Quad	$\left(Y^2_{1\times1}, C^0\right)$	$B : \{0, 1, 2, 3\} \rightarrow \{0, 85, 170, 255\}$
Oct	$\left(Y^3_{1\times1}, C^0\right)$	$B : \{0, 1, ..., 7\} \rightarrow \{0, 36, 73, ..., 255\}$
Hex	$\left(Y^3_{1\times1}, C^0\right)$	$B : \{0, 1, ..., 15\} \rightarrow \{0, 17, 34, ..., 255\}$

Table 2.3.1: We present the B mappings for luminance-only embeddings in order to introduce the Y^n notation as well as illustrate the corresponding luminance values embedded in a Cryptagram using that mapping for the embedding protocol.

(according to B) bits from m to successive CPBs color values, we fill the JPB from left-to-right, top-to-bottom, starting in the top-left of the 64 square pixel JPB, covering each channel independently. We can explore multiple color mappings B in order to see how q, p-Recoverability is affected by the $\left(Y^B_{1\times1}, C^0\right)$ CPB and B interaction.

We consider three mappings for B as shown in Table 2.3.1. The simplified representations for luminance will be used through this chapter. The superscript is the number of bits that can be embedded given the use of equally space values in $[0, 255]$, including extremal values.

Figure 2.3.3 illustrates the q, p-Recoverability of these choices. In comparing the best of binary, quadrature, octature, and hexature bits per pixel discretizations for the $\left(Y^B_{1\times1}, C^0\right)$ CPB, we have a sense of how the mapping choices perform relative to one another. Given a social network quality value (for JPEG recompression), we want to choose an embedding that allows for p very close to 1. If we choose the target quality to be 86%, then the values that are actually at $p = 1$ are the Quad and Bin mappings. These yield two and one bits per pixel, respectively. Because we are apt to conservatively choose a quality threshold assuming an OSN may lower their thresholds slightly (e.g., the OSN finds they save enough disk space without causing

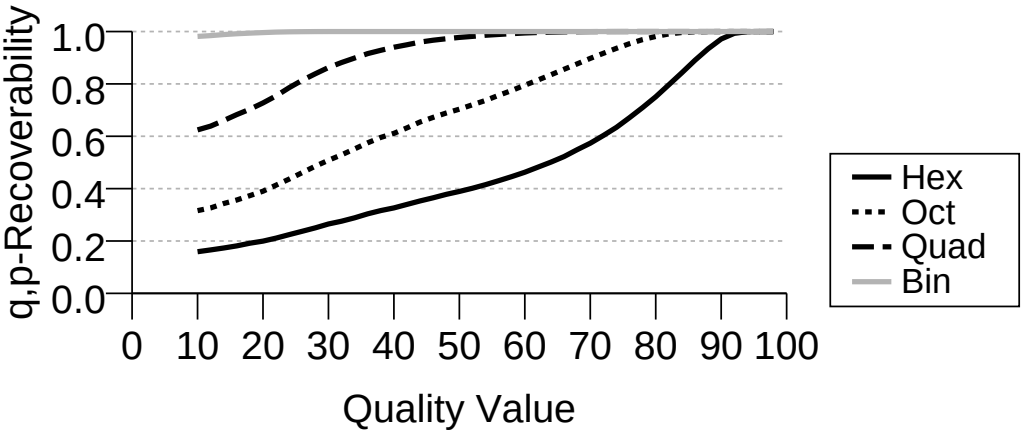


Figure 2.3.3: The relative performance of $\left(Y^B_{1\times1}, C^0\right)$ CPBs. We see that more discretizations results in weaker q, p-Recoverability as the quality to which we subject the JPEG to decreases. The tradeoff we must consider is what q, p-Recoverability we want to achieve (what minimum quality do we want a probabilistic guarantee) and how efficient we want for our embedding protocol.

user experience to suffer too much), we opt to use the Bin approach: $\left(Y^1_{1\times1}, C^0\right)$.

Figure 2.3.4 shows the results of our exploration of the chrominance CPB size and the impact of embedding in luminance and chrominance concurrently. We must use 2×2 CPBs in chrominance channels to embed one bit per channel's block (or a cumulative 0.5 bits per pixel gain). We can thus embed in chrominance as a function of the corresponding luminance values.[3] With this approach, we find that embedding more than two values per chrominance channel suffers low q, p-Recoverability. Thus while 4×4 appears to illustrate good q, p-Recoverability in the binary embedding case, we think that the efficiency gain is so marginal as to be negligible. Thus we consider the of use $\left(Y^3_{1\times1}, C^1_{2\times2}\right)$ CPBs to gain an additional 0.5 bits per pixel. This gain with chrominance always requires error correction in order

[3]Notably, if the luminance values are at the extremes (0 or 255), then we do not embed a chrominance value in that pixel since no valid chrominance value exists.

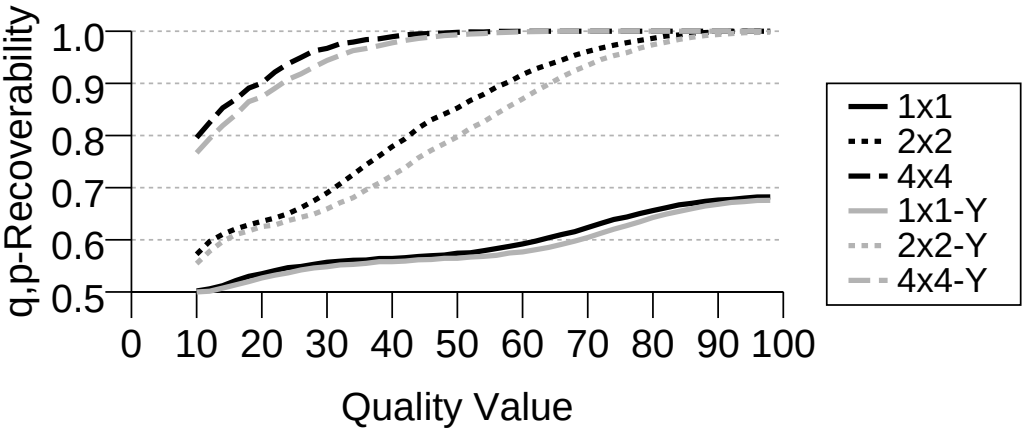


Figure 2.3.4: The feasibility of using chrominance to gain additional bits for embedding. All lines correspond to a chrominance B binary mapping. $n \times n$ corresponds to using only chrominance to embed bits using a binary mapping while keeping luminance set to 128. $n \times n$-Y embeds non-128 chrominance values along with chrominance. This plot also highlights the tension of using chrominance in tandem with luminance values. The diminished q, p-Recoverability may appear marginal when we are embedding binary data in the luminance space, but performance degrades significantly. As we explore the applicability of higher bit rates, we must carefully balance the interaction of luminance and chrominance.

to attain q, p-Recoverability that is robust to OSN recompression.

ON DECODING. To decode values from a Cryptagram JPEG, the decoder examines pixels in the same order as the encoder. Converting the RGB values to YC_bC_r, the decoding algorithm finds the nearest neighbor for the values in the co-domain of the B mapping for that protocol. The sequence of corresponding domain values is the decoded sequence of bits.

BALANCING q, p-RECOVERABILITY AND EFFICIENCY WITH ERROR CORRECTION

Since the nature of protocols that we investigate are probabilistically recoverable (the p in *q,p*-Recoverability), we consider the use of Error Correcting Codes (ECC) in order to improve the q, p-Recoverability of our protocols while maintaining efficiency of the embedding algorithm. Reed-Solomon codes are of interest to us for their well-understood ECC properties and space efficiency. In our case and especially in Section 2.5, we use $RS(255, 223)$ protocol, in which we use the $x^8 + x^7 + x^2 + x + 1$, or "0x187", field generator polynomial and 32 bytes for parity in order to recover up to 16 byte errors for a 255 byte transmission. The input to $RS(255, 223)$ is the encrypted output of the block cipher algorithm. With the application of $RS(255, 223)$ then, the q, p-Recoverable protocol directly embeds the output bit stream from $RS(255, 223)$.

From Figure 2.3.4, we note how the use of chrominance would always require ECC in order to recover from OSN recompression-induced errors for the CPB case we have highlighted: $\left(Y^3_{1\times1}, C^1_{2\times2}\right)$.

2.3.2 LOSSLESS COMPATIBILITY

Our effort focuses on the JPEG format given its online prevalence, but it's worth noting that our approach is seamlessly compatible with lossless formats such as PNG [33].

In this lossless scenario, we trivially achieve recoverability. The PNG format has a maximum per pixel efficiency of 64 bits per pixel. Each of the four channels, red, green, blue, and alpha, can store 16-bit values. We take 64 bits of sequential data in

m, split the 64 bits into four 16-bit segments, then write the respective 16-bit values into each of the four channels of a pixel.

2.3.3 Easy Key Management and Cryptography

Users have two options for managing access to their photos in Cryptagram: symmetric key cryptosystem or a hybrid (symmetric key and public key) cryptosystem.

In the case of the symmetric key cryptosystem, Cryptagram makes sharing keys easy. A single key can be used for an image, set of images, or an album, and shared amongst a group of friends. This makes key sharing easy and manageable by design, and our Cryptagram browser extension facilitates the use of a password across multiple images or an album by allowing users to store the password. Enabling a strong password to be applied across an entire album of photos means that Cryptagram makes key dissemination easy.

Employing a hybrid cryptosystem by following the principles of *key continuity management* [51] means that the Cryptagram design focuses on guiding the user to use a hybrid cryptosystem correctly. In particular, by (1) limiting the interface for the use of public keys for encryption and private keys only for decryption and (2) using strong defaults for the block cipher and public key cryptography algorithms, Cryptagram reduces the friction to secure and correct use of a hybrid cryptosystem.

For both schemes, users do not share the sensitive information through the social network. We advise users to use a separate channel (e.g., text messaging) to share sensitive credentials (e.g., an album password) so as to conform to the threat model in which Cryptagram is designed to protect users from a hosting OSN.

2.3.4 USABLE IMAGE IDENTIFIERS

While Cryptagram facilitates the creation of Cryptagram images, the question remains of how to identify and distinguish gray, fuzzy images for friends. We describe how we enable users to create images that are easier to identify for fellow human users.

TEXT WATERMARK: One challenge with the current format is that all output images look virtually identical. This is a problem when, for example, a user asks a friend for a Cryptagram password. Without a file name or album name, there is no codified way to refer to images. Using a simple extension to the encoding tool, we can enable the user to specify a text or image-based watermark to render underneath the Cryptagram image. A text watermark could specify useful identifiers, such as a URL or an email address for submitting password requests.

CHROMINANCE WATERMARK: In cases that we do not use the chrominance channels for data transmission, we can use these channels for identification purposes. We modify the C_b and C_r channels to add chrominance to output Cryptagrams and do so without corrupting the luminance payload.

We embed images in these chrominance channels so long as luminance remains unaffected. This watermark is not suitable for embedding most natural images since, perceptually, we rely heavily on luminance, but the technique works well with high-saturation icons or logos.

(a) Quality 78 **(b)** Quality 74 **(c)** Quality 70

Figure 2.3.5: Partial failure-resilient protocol.

2.3.5 Surviving Partial Failures

The current protocol has error correction and can withstand some degree of noise from JPEG but will fail with a cropping transformation. We can extend the basic design to provide cropping robustness by dividing a Cryptagram's payload into smaller units. We encrypt each block with the same password and decryption will involve individually decrypting and concatenating all blocks. If one block fails to decode correctly due to cropping, the integrity of other blocks and their sequence within the original JPEG remain unharmed. Such an approach, however, does not apply to storing arbitrary bit streams, but for images one can replace unrecoverable blocks with zeroes in order to display as much of the original image as possible as shown in Figure 2.3.5.

2.4 Implementation and Deployment

In this section we describe the current state of the applications deployed under the Cryptagram name, including several components and continuously evolving inner protocols. The code is open source and online:

http://github.com/prglab/cryptagram.

2.4.1 MICROBENCHMARKS

Cryptagram has over 400 active users (installed and currently present on the user's system) of its Chrome extension, distributed through the Chrome webstore. We request user-consent for an IRB-approved study to gather non-identifying log reports and consent has been granted from 373 unique browser installations.

We built the Chrome Extension with the Closure framework [54], requiring approximately 4000 Source Lines of Code (SLOC) [29, 129], porting the core components to a Firefox add-on with some additional code.

Our benchmarking framework consists of an ECC implementation and benchmarking code, and relies on the Reed-Solomon kernel module code ported to userspace, libjpeg-turbo codec, and a corresponding image interface ported from the Google Chromium browser [122]) (3000 SLOC).

The iOS App uses WebKit's JavaScriptCore to leverage the same cryptographic library as our JavaScript extension whereas the Android App achieves JavaScript integration through a WebKitView (2300 SLOC). These applications enable local Cryptagram encoding and decoding – we do not currently integrate with OSNs. While we do not have user data for the mobile versions of Cryptagram at this time to present, we have challenges in engineering and usability to consider. With respect to engineering we have found that configuring native cryptographic libraries to be compatible across languages can be a difficult sea to navigate: our wrapping JavaScript libraries results in a performance penalty but simplifies the assurance of algorthmic parity across platforms. We also encounter usability challenges with respect to accessing user's OSN photos from a third-party application. The current aim is to seamlessly integrate with a user's existing social workflow rather than

require users to use our product to access their OSNs. If Cryptagram were to be a self-sufficient entity, then we could foresee aiming to encourage user's to see Cryptagram as a portal to their OSNs. Of course, then one must balance the terms of service requirements of OSNs with the information our product reveals or does not reveal to those OSNs.

2.4.2 Browser Extension Decoder

We implemented the first version of the software as a browser extension, a framework supported by Chrome and Firefox browsers, which allows us to augment the user experience on any website by JavaScript injection.

For our first deployment of Cryptagram we adopted an embedding protocol with a $\left(Y^3_{2\times 2}, C^0\right)$ CPB. This protocol also embeds a checksum for verifying the integrity of the decoded encrypted bits. The checksum is not of the cleartext data; it is a checksum of the encrypted data and embedded adjacent to the encrypted data for data integrity purposes.

Decoding in place. Extensions can access pixel data of images on a website. The extensions perform image processing to produce new images to insert into the original image's container, as shown in figure 2.0.2. We add a contextual menu item so a user can right-click any image and attempt to decrypt it as a Cryptagram. With the correct credentials, the extension decrypts the original image, which pops into place.

2.4.3 Web-based Encoder

We wrote the web-based encoder in JavaScript with the Closure Framework, sharing much of the codebase with the decoder extensions. The encoder allows users to drag-and-drop cleartext images onto the encoder page. The drag-and-drop triggers an event to prompt users for a strong password (in the symmetric key case) as well as desired settings (e.g., the preferred tradeoff of a high-resolution, low-quality image or low-resolution, high-quality image). The encryption, encoding, and produced download zip requires no server interaction and thus allows for complete offline operation by end-users.

2.5 Evaluation

We now explore the evaluation of the Cryptagram system. We begin with microbenchmarks as well as observations that serve as background for the subsequent evaluations. In particular, we will present the efficiency performance of protocols that we find to be the most useful for end-users and reason about the utility of the current deployment.

2.5.1 Efficiency Microbenchmarks

With microbenchmarks, we aim to establish a sense of the tangible weight that Cryptagram adds to the user experience of sharing photos as well as the system overhead.

ON BROWSER PERFORMANCE. We found that the input file size to the encoder and decoder correlated linearly with time to execute. The approximate ratio of time to complete the operation (milliseconds) to input filesize (KB) was 2.684 for the encoder and 1.989 for the decoder on an iMac with 2 x 2.26 Quad Core processors in the Chrome browser (one core for the browser process). While the noticeable human visual reaction time is in the range of 190 to 330 milliseconds [68], the results demonstrate that the overhead of using Cryptagram for viewing OSN photos is marginal.

ON FILE SIZE. Since the high entropy of Cryptagram counteracts the compressive power of JPEG, the output file size depends entirely on the chosen embedding protocol and constraints imposed by the OSN. For Google+ and Facebook, uploading images have a cap based solely on image dimensions. The authors have found that the maximum upload dimensions in these OSNs is 2048×2048. This means that for a scheme that attains an efficiency of three bits per pixel, we can store at most 1.5 MB in the spatial domain of an uploaded JPEG image.

How does the size of the input data relate to the output Cryptagram image size? The nature of JPEG compression complicates this question. The output Cryptagram image may be saved at 100% quality, creating a large filesize footprint. While this may seem necessary given that we examine q, p-Recoverability with respect to the compression applied by an OSN, the composition of JPEG compression is neither idempotent nor cleanly-defined recursively. Instead, as we explore later in this section, we consider the observed error rates of compressing already-compressed Cryptagram images (simulating what an OSN would do).

	$(Y^3_{1\times1}, C^0)$			$(Y^1_{1\times1}, C^0)$		
Quality:	90	80	70	90	70	50
Expansion:	2.25	1.75	1.40	7.82	5.29	4.39

Table 2.5.1: We present the tabular data that illustrates the file size expansion when using various protocol choices in the Cryptagram framework.

Table 5.2.1 shows the expansion ratio from a given input size. For the case of a $(Y^3_{1\times1}, C^0)$ CPB with an output Cryptagram image with JPEG compression 70, the filesize on disk inflation is $1.4\times$.

For the sake of minimizing upload bandwidth, users may opt to export Cryptagram images with less than 100% quality and Cryptagram will still guarantee q, p-Recoverability within a certain range.

2.5.2 Compressing the Compressed

Apropos to the question of file size expansion, we examine the implications of a recompressed Cryptagram JPEG on q, p-Recoverability. Figure 2.5.1 shows the effects of exporting a Cryptagram to a JPEG Quality 1 and then (as an OSN would do) recompressing the image at JPEG Quality 2. The error rate indicates the fraction of CPBs that were broken through successive recompression. This data indicates that we can export Cryptagram JPEGs to 82% quality and OSNs' recompression still permits recoverability, assuming that we leverage $RS(255, 223)$ ECC.

2.5.3 OSN Photo Quality

As much of our evaluation relates error rates to JPEG quality level, we want to know the JPEG settings employed by popular OSNs. To estimate these quality levels, we

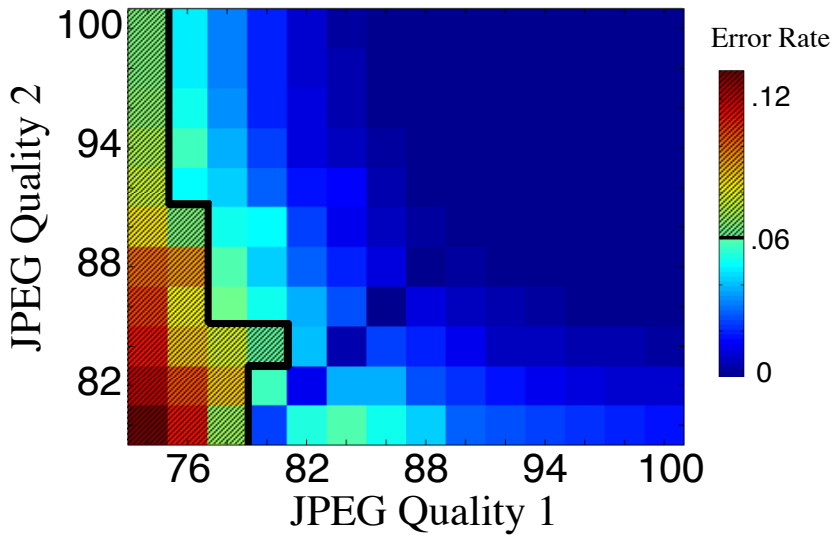


Figure 2.5.1: The effects of recompressing a compressed JPEG. The x-axis shows the quality of the original Cryptagram JPEG. The y-axis shows the recompressed quality of the JPEG. The line separating striped versus unstriped values is the q, p-Recoverability threshold we encounter with $RS(255, 223)$. Any values to the right of the 0.06 line show the successive recompressions that Cryptagram can tolerate for $\left(Y^3_{1\times1}, C^0\right)$. Error rates were determined by testing approximately 2,400 pseudorandom 8×8 images at each combination of quality levels.

exported a variety of images as quality 95 JPEGs, uploaded those images to both Facebook and Google+, then re-downloaded the images for analysis.

On Google+, 30 such test images came back bitwise identical, meaning images were not recompressed.[4]

Facebook, on the other hand, applies JPEG compression to save disk space. After downloading images from Facebook, we looked for evidence of quality in the JPEG headers. Out of 30 natural images, 25 came back with a JPEG quantization matrix exactly equivalent to that of a quality 74 JPEG, the other five having matrices equivalent to JPEG qualities in the range of 76 to 86.

[4]Google+ does recompress images for quicker display during album browsing but it is trivial to convert any such hyperlinks to their full-resolution equivalents.

Fortunately for Cryptagram, high entropy images all appear similarly to the JPEG algorithm and are treated predictably when uploaded to Facebook. All test Cryptagrams uploaded then downloaded came back with the quantization matrix from a quality 85 or 87 JPEG, which we measured by explicitly examining the quantization tables of the downloaded JPEG file. This quality level puts us safely above the necessary threshold of our deployed embedding protocol.

2.5.4 Embeddings with ECC

In this section, we examine the benefit of using ECC to reconcile the tradeoffs we must consider between efficiency and q, p-Recoverability. We presented in Section 2.3 the performance of the $\left(Y^B_{1\times1}, C^0\right)$ CPB for various B mappings. From that experience, we conclude that a protocol without error correction is limited to using quad or bin mapping strategies.

We examine the utility of applying our ECC algorithm of choice for embedding data to measure q, p-Recoverability in lower quality regimes of JPEG compression. With the use of $RS(255, 223)$ for ECC, we note that we embed 14% extra data for the recovery so our subsequent evaluation considers the effective efficiency of a system that adds this data overhead.

Figure 2.5.2 allows us to explore the design space of applying ECC to evaluate the q, p-Recoverability for given $\left(Y^B_{1\times1}, C^0\right)$ luminance-only embedding schemes. We see that the Bin, Quad and Oct embedding schemes perform above p=94 in the regime around 85%, thus enabling us to achieve q, p-Recoverability on Facebook.

Figure 2.5.3 illustrates the benefit of using luminance and chrominance embeddings in order to achieve 3.5 bits per pixel embedding efficiency for q, p-Recoverability

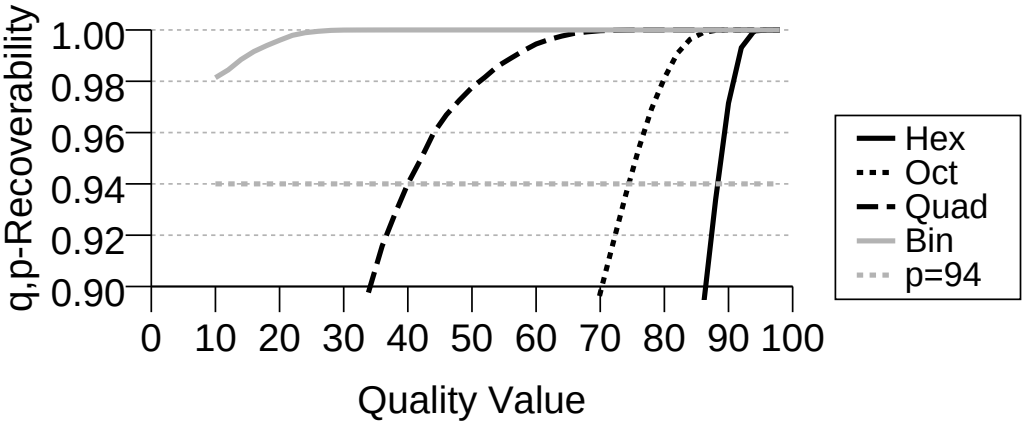


Figure 2.5.2: This indicates to us the feasibility of leveraging $RS(255, 223)$ to improve the q, p-Recoverability with various $\left(Y^B_{1\times1}, C^0\right)$ embedding protocols.

that satisfies OSN recompression and ECC. In the interest of saving space, we do not show the q, p-Recoverability curves, but instead summarize the details relevant to the ECC discussion in Table 2.5.2. Given that ECC with $RS(255, 223)$ recovers up to 16 bytes ($\approx 6.27\%$) of damaged bytes for every 255 bytes of data, we can establish our target recoverability probability at $\approx 94\%$; in other words, if less than 6% of bytes break then applying ECC enables us to use that particular encoding scheme. We highlight in Table 2.5.2 the q, p-Recoverable protocol that we choose for Cryptagram.

This efficiency is superior to X-pire! [11], which had a capacity of two bits per pixel with ECC. We have $1.75\times$ this capacity, significant considering the size and quality of images this enables users to upload to OSNs.

COMPARISON WITH STEGANOGRAPHIC EFFICIENCY. Though the goals of steganography and Cryptagram differ, both embed data in images, so we can com-

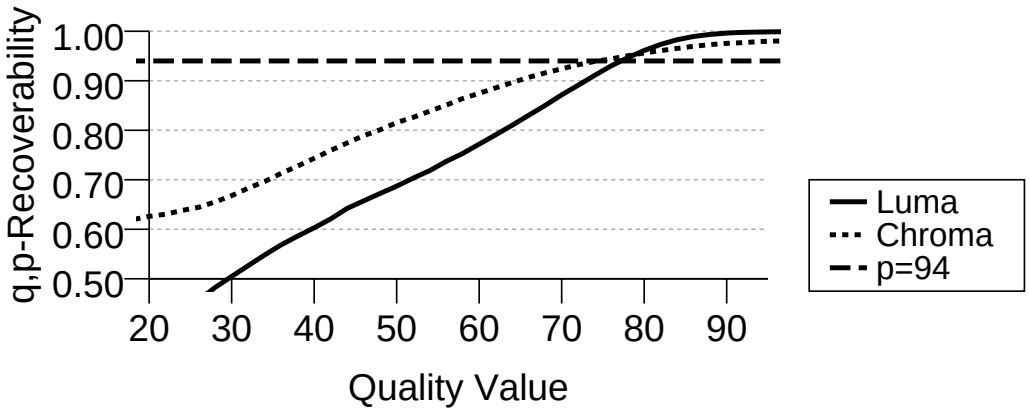


Figure 2.5.3: This indicates the feasibility of leveraging $RS(255, 223)$ to improve the q, p-Recoverability of a $\left(Y^3_{1\times1}, C^1_{2\times2}\right)$ protocol.

	Without ECC		With ECC ($RS(255, 223)$)				
lum $\lvert B\rvert$	q, 100-Rec (quality)	Efficiency (bits/pix)	q, 94-Rec (quality)	Effective Efficiency	Lum+Chrom q, 94-Rec	Efficiency (bits/pix)	Effective Efficiency
Hex	-	-	90	3.5	90	4.5	3.94
Oct	90	3	76	2.62	77	3.5	3.06
Quad	80	2	44	1.75	66	2.5	2.19
Bin	< 20	1	< 10	0.87	38	1.5	1.31

Table 2.5.2: Summary of the results that inform how to proceed with applying $RS(255, 223)$ FEC for embedding values in JPEGs that are recompressible.

pare the two in terms of bits/pixel efficiency.

Related work has expounded on the efficiency of steganographic embeddings [27, 128], reducing the approach to one embedding p message bits into $2^p - 1$ pixels, yielding a relative payload of $\alpha = p/(2^p - 1)$. While steganography choose slightly higher values of p a low value of p yields 0.42 bits per pixel for $p = 3$. In the highlighted row, our effective efficiency is 3.06 bits per pixel. In comparison, our approach represents a minimum $7.5\times$ improvement.

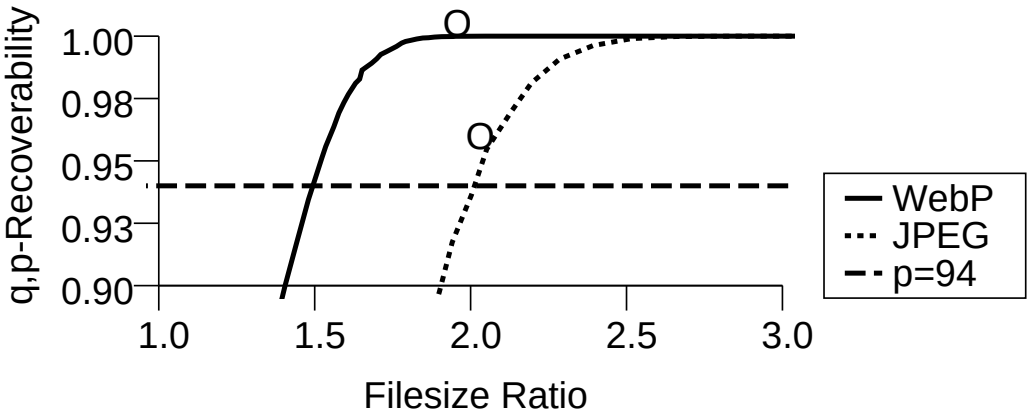


Figure 2.5.4: Showing the comparison of JPEG and lossy webp recoverability vs. filesize ratio. We draw the reader's attention to the $p = 94$ threshold as a point of comparison with the ECC studies in the rest of this chapter. We acknowledge that JPEG and webp quality settings are not related and cannot be directly compared. However, this figure shows that for a similar notion of q, p-Recoverability, webp has a smaller filesize expansion than JPEG to achieve the same probability of recovery. To note the distinction in the meaning of "quality" between libjpeg and webp, we highlight the points along the curves where quality is 74 for each codec.

2.5.5 File Format Embedding Comparison

In Figure 2.5.4, we show the q, p-Recoverability versus filesize ratio of JPEG versus webp image compression formats. By file ratio, we mean the on-disk size of the output image format for the same image canvas input. Notably, the embeddings are always three bits per pixel in the Figure. We see that for the same probability of recovery, p, webp has a much smaller filesize ratio than JPEG. As OSNs besides Google+ begin to experiment with webp deployment [113], the opportunity for lower bandwidth and storage requirements while maintaining q, p-Recoverability means that Cryptagram can be applied as improved media compression formats are adopted.

2.5.6 Deployment Usage Data

At the time of submission, Cryptagram has nearly 400 active installations, with 373 users agreeing to participate in our IRB-approved study. Through this study, we receive high-level data about the Cryptagram encryption and decryption habits of our users. The following data does not include the authors' own tests or images. We have had more than 3,300 Cryptagram image decryption events with more than 160 unique encrypted images generated. Of the decrypted images, we can confirm that 102 unique images have been decrypted from Facebook and 217 unique images from Google+.

2.6 Discussion

Applicability of q, p-Recoverability to Lossy Formats. OSNs continue to use lossy image formats in order to reduce demands on storage infrastructure and reduce delivery latencies to end-users. Recently developed formats shoud be considered given these goals. We have begun to examine the webp [55] format for Cryptagram. The tool of q, p-Recoverability applies in the analysis of these formats given that the spatial domain pixel value is the key component of Cryptagram communication.

Transformations. We aim to handle a variety of transformations with the development of q, p-Recoverable protocols. In previous sections, we discussed the design and evaluated our protocols' q, p-Recoverability with respect to the JPEG transformation. We have begun prototyping our approach to cropping and noising

transformations on images produced by Cryptagram as well, leveraging blocking algorithms coupled with ECC. While we do not address rotation explicitly, we do not consider such a transformation intractable as we apply techniques from QR Codes (two dimensional barcodes) by orienting corner features in future iterations.

Scaling transformations are of interest given the pervasiveness of lower resolution images (e.g., thumbnails) to partially depict images on a social networking website. We have considered the integration of pyramid representations [20, 28] in the design of future embedding protocols to meet this transformation request.

THE ECONOMICS OF PRIVACY. Our culture values greatly the power of images to document and record in ways that words simply cannot. We say *seeing is believing*. Images convey a range of human experience, and unfortunately, that includes images that can irrevocably damage a person's reputation.

OSNs offer privacy features and third parties have even developed commercial products to address photo privacy. McAfee Social Protection lets users store cleartext photos on their server while uploading blurred versions to Facebook, then facilitates access requests [111]. This superficially addresses photo privacy, but in the end, amounts to an escrow service that redirects trust from one third party to another.

Our optimistic vision for this project is that its adoption could articulate to OSNs that users desire increased ownership over personal data. We envision a scenario in which an OSN embraces the philosophy of Cryptagram and provides client-side tools to make end-to-end encryption feasible. Wide adoption of Cryptagram would require more storage for the encrypted files and may create less potential for targeted advertising.

3

Lightweight Private File Sharing Across Federated Services

Millions of users share files online with popular file sharing services [5]. The convenience with which users have gained access to the cloud for leveraging the availability of their files has been a focus of file sharing designs in recent years. Seamless integration with the user's file browsing experience has made the cloud's potential available to the masses.

The need for privacy while sharing files remains a crucial topic. MegaUpload has led the publicity effort in offering a file locker and sharing service that ensures uploaded files are immediately encrypted before any bits of the original file hit the

wire. Efforts by Silent Circle have demonstrated a growing need for accessible and usable privacy enabling technologies [24, 50].

But the widespread adoption of transparent file sharing services such as Google Drive and Dropbox has followed the trend of popular web-based successes: the afterthought is privacy [93]. While these services take measures to ensure a compromise of one's machine does not affect the integrity of access to files, the question remains of how much to trust cloud storage providers with our data. In essence, users want an application that retains the convenience of sharing files (a familiar interface for interacting with files) while also providing end-to-end privacy for their data without drastically inflating the cost of the service or storage.

This chapter presents Lockbox, a system that explores the design points of providing usable privacy (data confidentiality) and lightweight impact on untrusted cloud provider services. In particular, the challenges that we face in Lockbox involve reconciling the following objectives:

- **Cryptographic Privacy Guarantees:** Leveraging vetted cryptographic primitives to enforce access controls means that we must design Lockbox to accommodate a difficult balance of convenience and correctness with the chosen strength and combination of cryptographic primitives in the file sharing system. We demonstrate the use of a hybrid cryptosystem in order to provide both strong privacy guarantees and (with the integrated key distribution network) a convenient means of handling key distribution and management.

- **Lightweight File Footprints and Versioning:** Lockbox mechanisms for storage must account for minimizing storage costs both on the server and end-user devices while retaining the useful features of such approaches in order

to achieve these goals. We accomplish this by separating the techniques for managing versions in the cloud and on end-users' devices. Is Section 3.2, we introduce the protocol that enables interoperability between a delta-based cloud store and a local object blob store when operating over encrypted data.

In the course of providing file backup, modern services have enabled users to retain versions of their files. This feature allows for users to potentially rectify mistakes as well as review how ideas have evolved. Nevertheless, previous work has not explored the use of versioning on encrypted data with the aim of minimizing the file system footprint in the cloud to the best of the authors' knowledge. Lockbox also demonstrates an API that allows for programmers to specify file-specific plugins that implement more efficient versioning schemes for the files that users edit.

- **Detecting and Recovering from Server Equivocation:** Malicious server administrators may choose to attack the service by *forking* the view that multiple user have of (supposedly) the same file. SPORC [42], BFT2F [74], and Frientegrity [43] explore how to design systems that enforce fork and fork* consistency. These past works assume that users will easily collude in order to resolve fork inconsistencies in the case of a malicious server. For non-technical users, peer-to-peer interactions may present too high a barrier to entry. Lockbox asserts however that multiple independent servers can satisfy the security requirements of privacy-conscious users through a two-party model. Lockbox introduces the use of a two-party model for users to authenticate the correctness of transactions through an untrusted cloud provider to verify fork consistency.

We use a federated system of independent servers to achieve multi-party-backed security guarantees in the design of Lockbox. We separate the key management and file storage services into two separate cloud providers. Data verifying the consistency of one service is sent through the other in order to detect and prevent malicious server-side behavior. With our multi-party system and end-user storage design, we illustrate how Lockbox recovers from fork* consistency attacks.

- **Key Access and Revocation:** Users should have clear control over who has access to their shared files through Lockbox. We describe a social network to facilitate generated key distribution for users of Lockbox. The benefits of the network are two-fold. The first is that a trusted service for distribution serves as alternative means of verifying the keys of trusted identities in the network. We cover the second point with the following objective.

- **Usable and Convenient Cloud Storage:** We aim to make possible the use of cloud-backed services for end-users as seamless as possible, even when the interface is not a web browser but our own frontend. With the key distribution network integrated with the Lockbox service, Lockbox leverages the security properties of an underlying hybrid cryptosystem and users encounter fewer of the details of public key cryptography. In particular this means that a user of the network is less likely to misuse the system, keys, or network to her privacy detriment. We have also designed the system such that users may use different storage services (even dramatically different designs) for the storage needs, given the same Lockbox client. We demonstrate the porting of our

backend to use AWS storage services entirely without running a proxy with Lockbox-specific logic.

Achieving all of these goals simultaneously has been an elusive task. Existing solutions sacrifice one design point for the assurance of others. By using TrueCrypt on top of Dropbox [10], users must sacrifice granularity of control (one key for everything) and store the whole file (no deltas) for every encrypted revision uploaded to Dropbox. In systems like Eyo [119], a design for transparent storage is presented but without considering the challenges of sharing with multiple users or ensuring correct, end-to-end private data backup or sharing. SPORC aims for lightweight private concurrent access to cloud data for data types in which Operation Transformations can apply [42], but without considering generic file types, key distribution, or explicit versioning control by end-users.

To the authors' best knowledge, Lockbox is the first system to offer lightweight storage that ensures user privacy with both usability and cryptographic values retained. We have implemented Lockbox primarily in C++ as a desktop client. We have also engineered reference implementations of the Storage Service Provider. We demonstrate that while Lockbox's time performance is an order of magnitude slower than local git client and server operations, we do outperform Dropbox on our client machines. With respect to server space requirements, we show that we are within a competative multiple of on-disk space usage compared to a default git client and server installation (no encryption/decryption). We have open-sourced our code here: `http://github.com/prglab/lockbox`.

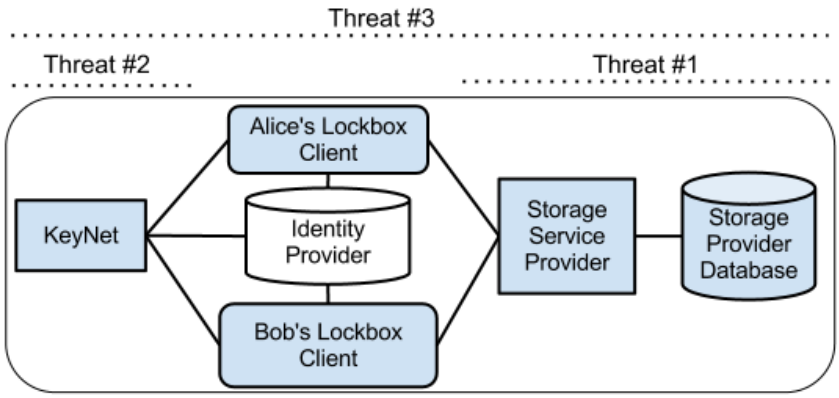


Figure 3.1.1: High-level overview of the components of the Lockbox system as well as a guide for the threats that affect the design of the Lockbox system.

3.1 Security Overview

In this section, we examine the threat model that Lockbox aims to address with its system design as well as address why a popular solution to the idea of convenient file sharing is inadequate and misleading with its security properties.

3.1.1 Threat Model

Figure 3.1.1 shows an overview of Lockbox as well as the threats that we consider most relevant in the discussion of this system. Lockbox aims to address the following threats:

Threat #1: Cloud Storage Provider Compromised.

In this threat, Lockbox guards against a curious database administrator or other external attacker who has access to the data stored in the databases of the storage provider. Our goal is data confidentiality, not integrity or secrecy. We assume the

adversary may cause users to have forked views of what was originally the same data. Moreover, we assume that the service provider may forge keys for access data or brute force session key deciphering. With the increasing dependence on cloud services, this threat is increasingly important.

THREAT #2: KEY DISTRIBUTION NETWORK COMPROMISED.

In the case that attackers compromise the key distribution network, users may be at risk of encrypting data for unintended recipients of their data. With the network identities compromised, the challenge becomes the discovery of true identities in the system.

LIMITATIONS.

Obfuscating sensitive or important data access: Data access patterns reveal information about the relative importance of documents on a filesystem. While hiding the existence of sensitive data (e.g., queries in TrackMeNot [61]) concerns some we believe that this outside the scope of Lockbox.

Copy and Paste: When a trusted user receives shared data through tho Lockbox service, the user may copy the contents of the data outside of the scope of the Lockbox service. We call this malicious behavior (from a, therefore, wrongly trusted user) a "copy and paste" attack. Since Lockbox must be a small piece in the broader ecosystem of computing devices, the risk of a copy and paste attack is inevitable and a threat to the expectation of data confidentiality that users have of such a system as Lockbox.

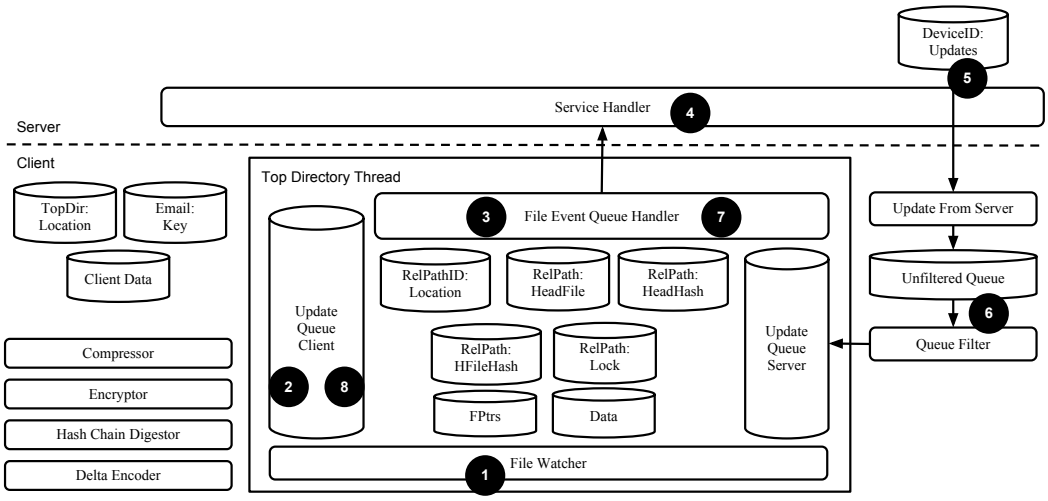


Figure 3.2.1: The Lockbox Architecture.

3.2 LOCKBOX DESIGN

Lockbox aims to achieve the following: strong privacy guarantees, lightweight file footprints, file versioning, key access and revocation, and usable, convenient cloud storage.

In this section, we discuss how the overarching design of Lockbox meets its end-user goals and then examine how we achieve the aforementioned design points in the architectural details of Lockbox.

3.2.1 CLIENT-SIDE OVERVIEW.

Figure 3.2.1 illustrates the architectural overview of the Lockbox end-user system. To see how Lockbox meets its end-goals, we walk through the steps that illustrate the actions of the client and its interaction with the server (the numbers in the following steps correspond to markers in Figure 3.2.1.

1. The File Watcher within the Top Directory Thread monitors watched directories for file changes: additions, modifications, and deletions. When an event triggers an update, the File Watcher posts an update to the Update Queue Client database. Notably, the Top Directory Thread is instantiated once per non-overlapping, top-level directory that the user intends to synchronize through Lockbox.

2. The Update Queue Client is the queue of file events that have been triggered by file system activity. This queue is ordered based on time.

3. The File Event Queue Handler manages most of the details of the interactions between the cloud and the local file system. In particular, the File Event Queue Handler ensures that for a local file update that a renewable, timed lock on that file has been set through the service provider. Any updates are then pushed to the cloud (state updates, file uploads, etc.).

4. The Service Handler at the cloud service provider requires a minimal service interface. The Service Handler primarily manages the ingress and egress of state and packages.

5. The Service Handler offers a polling interface for updates to end-users. By mapping a globally-unique identifier (GUID) to the queue of updates for a device, polls from end-user devices can be returned with the summaries of updates that that user device needs to synchronize.

6. To ensure the persistence of data, the Update From Server thread persists lists updates to an unfiltered queue before acknowledging the reception of the

updates to the cloud. (Once the updates have been acknowledged for reception, the server is free to delete the entries from storage/memory.) The Queue Filter thread reads through the Unfiltered Queue and moves those updates to the Update Queue Server for the appropriate Top Directory thread.

7. The File Event Queue Handler prioritizes reading updates from the server. When a queue update in the Update Queue Server database has been detected, the File Event Queue Handler executes the appropriate actions to bring the local client up to the latest view.

8. Since most of the interfaces for file system monitoring offer limited filtering capabilities, file update events caused by Lockbox synchronizing from the cloud are filtered to avoid an infinite loop of file "updates."

The only components not presented are the exchanges of hash chain digest histories that allow users to detect server equivocation. In the Lockbox model, these exchanges occur through an entity that does not collude with the server: directly between clients or through another server. This multi-party model for exchanging hash chains empowers Lockbox users to minimize the incentive for Lockbox servers to lie about the messages that can be exchanged through the Lockbox system.

3.2.2 Lightweight File Versions and Privacy

At the core of Lockbox is a system for storing files for versioning and backup purposes. We discuss how Lockbox monitors files and converts copies of those files to *objects* within the Lockbox system.

```
enum PackageType {
  SNAPSHOT,
  DELTA,
}

struct HybridCrypto {
  1: required string data,
  2: map<string, string> user_enc_session,
  3: string data_sha1,
}

struct RemotePackage {
  1: required string top_dir,
  2: required string rel_path_id,
  3: required PackageType type,
  4: required HybridCrypto path,
  5: required HybridCrypto payload,

  # Contains the SHA1 hash of the delta's
  # previous whole file hash; i.e., the
  # hash of the file that this delta should
  # be applied to.
  6: required HybridCrypto delta_prev_hash,
}
```

Figure 3.2.2: Thrift code that demonstrates the components of a package that is exchanged between clients and the server in the Lockbox architecture. `top_dir` and `rel_path_id` correspond to the GUIDs that have been assigned for the Top Directory and the Relative Path within that Top Directory. The human-readable values that these GUIDs correspond to are only visible to trusted clients.

Filesystem monitoring in Lockbox occurs at the granularity of a directory. Users may monitor multiple, non-overlapping directories on their filesystem as well as apply unique sharing settings to directories even distinct settings to directories in the same monitored hierarchy (see Section 3.2.5).

Users set watchers on specific directories through the Lockbox frontend. When Lockbox watches a directory, Lockbox synchronizes the directory with the user's storage provider. Lockbox also synchronizes any subdirectories and files with paths relative to the synchronized parent directory. This means that one cannot separately sync subdirectories. Users may set different access controls per the directory structure.

The watcher service generates events based on changes to local files. For the moment we focus on the case of a new file saved to a monitored directory.

The File Event Queue Handler calls the appropriate handler method to upload the side effects of the event. That method has the purpose of creating the appropriate RemotePackage to upload to the Server. Whether the upload must be a delta encoded value or snapshot, the contents are encrypted following a hybrid cryptographic scheme (`struct HybridCrypto`) consisting of the session-key block cipher encrypted data (`data`), hash of the encrypted data (`data_sha1`), and the public-key encrypted session keys (`user_enc_session`).

NEW FILES ("ADDED" EVENT)

When the File Watcher detects a new file in a monitored directory, Lockbox reacts by first determining if it should prepare the file for upload to the cloud and if so, do so. The local database maintains a mapping between the local relative path and

the object. When a user adds a new file, the application must check if another user sharing the directory has registered the hash of the relative path on the storage service. For the moment, assume that the file is new (we will discuss the scenario where the local client detects a file locally and remotely with the same path but not yet synced across the network) so we find that no hash of the relative path.

Lockbox names the file based on a GUID received from the cloud. For that relative path within the Top Directory, that Relative Path GUID will represent the file

MODIFIED FILES

When Lockbox detects a modified file, the client must ensure it can consistently sync the data with cloud, determine the most efficient way to express the change through the cloud (delta or snapshot), and then execute the appropriate algorithms.

THE CLOUD LOCK. To support concurrent writers in Lockbox, we address how we manage lock state. The client attempts to obtain a lock if the user has a file modification to upload to the cloud. The client requests the lock through the storage service provider. The lock returns with a status data, a timestamp, number of seconds until the client must renew the lock in order to retain the lock, and (optionally, as this information could be locally cached) the users for which the file should be encrypted. The contents of the status data indicate whether the user holds the lock. If the data contains nonce data, then the client signs the value and sends it to the storage provider in order to lock the data with a signature. The timestamp and expiry length indicate when the central system locked the path as well as how long the client has to lock the data for upload.

If another has already obtained the lock but the user does not renew it or upload the data in the specified time limit, the local user downloads the updated delta, reconciles the diff manually with the locally changed version, then goes through the process of obtaining a lock and sending an update.

DELTA ENCODING. Assuming that the local client obtains the lock, the client uploads an object that corresponds to the next revision of the file. Beyond text files, many delta encoding schemes have been developed to ensure minimal data transfer for data. Amongst the available protocol implementations, we consider rsync [124], courgette [56], and bsdiff [100]. Using the Delta Encoder interface, Lockbox computes the difference between the latest change to a file and the corresponding object in its local object store. It is this delta that is the data that Lockbox encrypts and uploads to the cloud. The naming convention follows as above with new objects.

3.2.3 LIGHTWEIGHT SYNCHRONIZATION WITH OTHER USERS

We elaborate how to synchronize data with other users in the Lockbox system with the aim of achieving a lightweight file footprint. We assume for the moment that we have loaded trusted public keys into Lockbox so that when we choose to add or update data in shared directories, no additional key setup is required. We elaborate the key setup process in later sections.

LOCAL UPDATES

This scenario corresponds to the aforementioned local case described in the previous subsection. The user agent (client) locks the relative path in the cloud. The

proxy timestamps the lock and future visitors may find that the lock has expired if a transaction associated with the lock extends beyond $t_{timeout}$ seconds. The user agent renews the lock in case a large file takes beyond $t_{timeout}$ seconds to upload. Once the client uploads the object, it releases the lock.

We note that when uploading the pubkeys object, the user references the proxy's association with permitted keys for that object so that clients encode updates for the correct identities.

REMOTE UPDATES

We have discussed how Lockbox monitors local file changes and then uploads updates to the cloud, but we must also find a lightweight mechanism to receive updates through the cloud from other users in Lockbox.

When the proxy detects that an uploaded modified file, the proxy either sends an event to the local client or the local client polls for the update, the application decrypts and decompresses the contents of the previous version of the file, computes the delta between the current and previous versions, then writes to the user directory the updated file.

To facilitate quick discovery of new notifications, the service maintains mappings between the user ID and objects as well as the triple (UID, private key, device ID) in so that when a triple requests updates, the system returns values corresponding to the objects that the client should fetch from the proxy to update its corresponding local file objects.

CONFLICT RESISTANCE AND RESOLUTION

Two important scenarios emerge when discussing conflict resolution in the Lockbox scenario: new files or modified files. In the case that two devices created a relative path file with the same name, then the contents of the file will need to be synced through the cloud, reconciled by one of the users, and the reconciled version uploaded.

In general, this means that the first writer gets the lock but that the last writer has the final word. The clients can detect these concurrent changes and prepare the documents so that the user will see that the conflict has occurred. The client knows to upload only when the user has reconciled the documents by merging changes into the original filepath. If remotely users continue to change the file while the user is interacting with the document, then the client will present these updates as further conflicts (copies of the new versions) to the user so that the user can manually reconcile these conflicts.

3.2.4 USABLE KEY MANAGEMENT

We introduce a service by which we enable users to leverage aspects of public key management without the encumbering overhead of understanding the underlying protocols [114, 130]. We call the web service KeyNet. KeyNet allows users to seamlessly upload and share public keys, reducing the friction to bootstrapping use of public key cryptography. As a web service, KeyNet accepts uploaded keys from authenticated identities (OAuth, verified email addresses). Users set privacy controls for uploaded data. According to the privacy controls, KeyNet makes the data searchable to facilitate lookup. While KeyNet offers features like many existing

key servers, its API for tight-integration with an application follows more closely what has been a design idea from Garfinkel [51].

We couple KeyNet with Lockbox in order to facilitate the use of a public key infrastructure to ensure the security of files shared through the Lockbox system. When users generate private and public key pairs is Lockbox, they have the option of verifying an email account with KeyNet in order to associate their keys with the email address. This use of a verified identifier allows us to trust the use of the email address (or, the user ID) in KeyNet with the public key fingerprints that the user uploads to the service through the client. Optionally, users may also offer a human readable name to facilitate searching within the service for new friends. Searching may also be performed on specific public key fingerprints. Users may also choose to not allow their identifiers to be searchable except by specific user IDs.

The queryability of understood identifiers such as human-readable names and email addresses allows users to discover friends through KeyNet. Similar to public key servers, these mechanisms enable users to install keys that they trust to their application without the overhead of understanding the intricacies of public key cryptography.

Lockbox encourages users to use one private key across a range of devices. To facilitate private key transfer, we encourage the use of KeyNet to create an symmetrically encrypted package given a user-supplied password to transfer and give access to the file by the new device through encrypted transport channels (e.g., TLS). Lockbox attaches an expiration date to the package. In this way, users can have secure, temporary storage of their data.

AUTHENTICATING ACCESS FOR ALL TRANSACTIONS.

We verify the identity of a valid key for a particular directory by using a cryptographic nonce to authenticate keys used to upload data to an account. This feature defends the system against replay attacks as well as attempts to upload data from an unauthenticated source.

In the case that we use a service like Amazon Web Services, we rely on their access control infrastructure to moderate access to data based on the access and secret keys configured by users through their Lockbox client.

ALL NEW KEYS FOR A USER.

In the case that a user must invalidate all of her old keys in order to update the system with new ones, Lockbox requires that other users be involved for the recovery to be complete. In particular, any user that then has access to the files and is available to modify files is notified of the request and the client creates new public key encrypted values to upload for the user. This ensures that up-to-date and correct mappings of public key access to files. The drawback of this approach, which is at the core of Lockbox, is that we rely on other users in order to recover data that may not be accessible to us at a new machine or ID. This ensures however that Lockbox satisfies the principle of least privilege so that unauthenticated users are not permitted to gain access to files that they do not have at their disposal.

3.2.5 ACCESS AND REVOCATION CONTROLS

Users use the Lockbox frontend GUI to assign email addresses to access directories and files synchronized with the Lockbox system.

ADDING USERS.

When an owner adds a user to the Lockbox client keyring, the user must attach the public key for the user (obtained through a trusted channel) or rely on the KeyNet service in order to share the file. Through the UI, the user can assign a user to have read or write access to a file or directory. When the user changes the access permissions, then the client must update the service proxy. For the directory, the user must upload keys for that user so that the path keys object has the correct mapping for access control. The mappings maintained at the service proxy govern access control to the path objects in the Lockbox system.

Users can add new user by searching through KeyNet for the email address with which they want to share data. Finding a correct match for a friend, a user may import the friend's key from the KeyNet. To end-users, this simply appears to "add" a friend.

REMOVING USERS.

When a user must be removed from a path, a manager of the file removes the user's ID with the client UI. The client subsequently removes the user's ID (and corresponding keys) from the pathkey object for the corresponding objects. This removal ensures that all new versions of the file created by users respect the permissions set by the latest managerial change.

RE-ADDING USERS.

When a user is removed from access to an object and then re-added several deltas later, the question is: should that user be granted access to all of the deltas that he

was intermittently not granted access to? Lockbox enables both choices but defaults to the following case: Lockbox only grants access to the most recent delta and not the intermediate ones that the user was not granted explicit access to.

In that case the use of another user's client is necessary (and this seems likely to be possible if the permissions for files has been adjusted through a client). Given the readmitted user's key and last accessible object, the client is able to look in its own history for the object value. The object is then reconstructed from the history up to that point and then diffed with the most recent version to produce the snapshot for readmitted user. Access to this object is made available all users but is explicitly labeled as readmit. This allows the user to find the file and use that for the delta that affects them.

In the case that a user is granted managerial access to the path in the future, then the full history a file is revealed the user. While this computation requires computation at the end of the access granting client, the once readmitted user, now manager can also provide readmission access to users.

Invalidating Keys and the Implications on data history.

Users determine if an invalidated key should have access revoked in the history of all objects or merely for future encryption. The proxy executes the computation for the former, if the user deems it necessary, by using the key to object mappings. When a key has been compromised, we assume that sensitive data associated with that key must be prevented from being accessed by users other than those that have the proper access. The cloud storage provider ensures that the compromised private key is not used to authenticate access to any of the data. In a token sent from the

```
struct HashChainDigestHistory {
  1: required list<string> history,
}

struct HashChainDigestEntry {
  1: required i64 view,
  2: required i64 seq_no,
  3: required string digest,
  4: required string signature,
}

struct VersionInfo {
  1: map<string, HashChainDigestEntry> rpid_hcd,
  2: map<string, HashChainDigestHistory> history
}
```

Figure 3.2.3: Thrift structures exchanged for detecting server equivocation using hash chain digest histories.

client to the reference Lockbox service, the invalidated key that signs the data will be rejected.

3.2.6 Detecting Server Equivocation

Lockbox clients exchange messages through a remote service to verify operations of a distinct, non-colluding service. That two-party model for sharing information ensures that operations that are exchanged through the server are delivered from the server as expected. For instance, Lockbox clients use the KeyNet service to exchange hash chain digests, which reflect a fork consistency check on the behavior of storage server.

Recall that the hash chain digests are computed from the successive hashes of

concatenating the hash of the current contents with the previous hash:

$$HCD_{i+1} = hash\left(hash(contents) \cdot HCD_i\right)$$

Clients exchange hash chains of the packages that are transmitted, which includes the hash of the stringified packages. Storing these messages and comparing histories as new files arrive enables the direct detection of the existence of misbehaving nodes, assuming non-collusion between the parties through which the hash chains are communicated.

The multi-party allows the exchange of hash chains through multiple parties (repeated work but redundant checking) to aim for Byzantine Fault Tolerance [74]. This use of hash chain digests exchanged through third parties that do not collude with the storage provider enable the detection of discrepancies in the behavior of untrusted cloud storage providers.

When a fork has been detected, users leveraging their current file versions snapshot the current file and upload the hash to the new, trusted service provider. They thus proceed as normal using this new service.

3.3 IMPLEMENTATION

We built the Lockbox client principally as a desktop application. We have included basic reference implementations for the cloud storage interface as well in order to minimize costs in our evaluation as well as easily bootstrap any user willing to host their own Lockbox repository. The source code for Lockbox is available here:

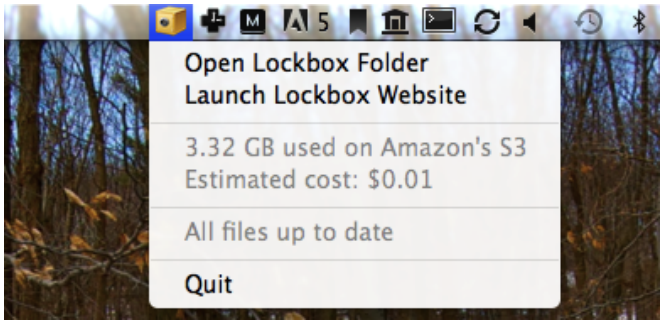


Figure 3.3.1: In our aim for seamless convenient integration for end-users, we show our status menubar item which allows users to see relevant information for their storage needs in the Lockbox system.

`http://github.com/prglab/lockbox.git`.

3.3.1 Client Implementations

We have implemented two versions of Lockbox, one in C++ and one in Python. While most of the design of functionally overlap between the two implementations, the C++ version was developed following the Python one to control better for memory usage and cryptographic operation interoperability and runtime.

C++ Client

The C++ codebase consists principally of 5300 lines of C++ code built through autotools. The codebase relies heavily on code ported to autotools from the Chromium project for base library support (100K lines), leveldb (18K), cryptography interfaces (12K), and binary delta encoding with courgette [56] (7500). We have implemented the network communications using thrift's [38] (version 0.9.0) serialization and non-blocking server implementations. We managed all of databases at the client and server using leveldb [30] (version 1.10.0), emphasizing the facility to scale both the

number of server processes as Lockbox scales and for clients that wish to sync many directories from a single machine.

Python Client

The Python desktop application consists of 5,400 SLOC of Python code. We use M2Crypto for all cryptographic operations. Additional packages that we rely on for enabling features of Lockbox include: boto, Mako, nose, PyYAML, simplejson, and watchdog. Lockbox stores all data that must persist across sessions and transactions in a SQLite database. For the Mac OS X application, an additional 200 SLOC are used to bridge the native GUI environment on Mac OS X with the Python core using PyObjC [97]. Ultimately, we aim to facilitate users sharing files without worrying about an eye in the sky as easily seeing their data as sharing is encountered today. To that end of facilitation, Lockbox aims to make the file sharing and backup experience as seamless for end-users as possible.

At the core of the Lockbox system's user-side code is the ability to detect file changes across multiple directories as configured through the Lockbox client. For Python, the watchdog module has served to provide a cross-platform interface for detecting file modification events [7].

The web application as been prototyped also in Python leveraging the Tornado Web Service platform from Facebook [39]. We separate the notion of the reference storage service implementation from the key distribution network. The key distribution network assumes, also written with Tornado, ensures that user accounts have been authenticated with OAuth2 so that identities associated with an email address can be trusted, assuming that the email account has not been compromised.

3.4 LOCKBOX PRIVACY GUARANTEES

In this section, we discuss how Lockbox addresses the threats outlined in Section 3.1 given the design of the system from Section 3.2.

3.4.1 CLOUD STORAGE PROVIDER COMPROMISED.

Lockbox aims to protect data confidentiality against the threat of a compromised cloud storage provider. We consider two types of cloud storage provider compromises: breach of confidentiality and server equivocation.

In the case of a breach of confidentiality, we consider the case of the curious database administrator or unauthorized access to users' data on the storage servers. Lockbox leverages a hybrid cryptosystem to store encrypted data on the cloud storage provider. End-clients perform the computations for key generation and encryption. We assume that users trust their end-host devices. In order to encrypt data with the hybrid cryptosystem, the users must leverage either their own means of ensuring that they use verified keys in their system or the KeyNet distribution service.

Server equivocation occurs in the threat model of fork consistency: the server "forks" its users so that different users see different versions histories or sequences of changes to the "same" file. Lockbox uses hashed chain histories in order to compute and compare the integrity of the data shared. Moreover, while these histories are shared in-band, Lockbox also enables users to use the KeyNet service to test for server equivocation. This separate channel and second party protects users in the case of fork attacks by the cloud storage provider.

GUARANTEES. More intuitively, Lockbox ensures that the following property holds: Sensitive data is never available in plaintext at the storage server. Lockbox thus thwarts curious database administrators and passive observers.

Finally, we believe that the passive attack model is realistic because malicious database administrators are more likely to read the data, which may be hard to detect, than to change the data results, which is more likely to be discovered once users compare notes. For instance, chaining keys can be used to ensure that the storage provider cannot insert a malicious change to the data.

3.4.2 KEY DISTRIBUTION NETWORK COMPROMISED.

We now describe the second threat where adversaries compromise KeyNet and address the results of any computation based on the falsified keys. The consequences of a breech in the key distribution network cannot be solely addressed by relying on the underlying cryptographic mechanisms since adversaries may use the network to disseminate false key identity information. In particular, adversaries may mislead users to encrypt data for an adversary who has swapped the keys with an otherwise trusted identity.

The use of the trust network existing work to establish trusted relationships ensures that users encrypt data for identities that they have already confirmed as trusted in the network. This is where the traditional notions of a web of trust enter into play. The use of separate channels to confirm the identity of keys that the user has added to her trusted keyring for a users Lockbox instance.

The core of the problem stems from the use of compromised keys from the beginning of any transaction. This means that if the users begin to use the KeyNet

before an adversary attacks, then the users will have trusted keys in their keyrings. This means that the users of the system can detect unexpected updates in the system of user keys. When conflicts between key fingerprints emerge, then Lockbox notifies users of this problem and asks them to contact their friend through a separate channel in order to ensure the correctness of their public keys.

4

Realizing Contextual Integrity in an Online Social Network

Privacy and sharing are at odds in online social networks [57, 59, 134]. Privacy controls and settings on Google+ and Facebook are far from perfect. Johnson et al. demonstrated the disconnect between Facebook privacy controls and the settings that users believe they have set for their accounts [64]. While Google+ offers a different privacy settings mechanism ("circles"), Kairam et al. [65]'s analysis argues that Google+ users desire privacy controls to encompass factors that are not explicitly apart of the circles construct; in other words, circles lack expressiveness.

Moreover, many examples of online social network privacy failures pervade the

modern landscape. Due to poor engineering practices, Facebook deployed flawed code that resulted in their CEO's private photos being leaked due to a bug in one of the "reporting flows" on the site [35]. In several court cases, users expectations of privacy have been unmet leading to the citation of evidence gathered on Facebook for a growing number of divorce proceedings [26, 117]. Even in casual settings, users have been unable to concurrently share their revelry with friends as well as maintain the standards expected of them by employers when representing themselves online [80].

Thus, the current privacy setting models in OSNs have three basic design flaws: (a) there is often a mismatch between user-specified settings and the user perceived sharing intents; (b) those models offer inadequate privacy protection to the users; and (c) the systems upon which they are built do not verify the user's intentions.

We argue for a completely new model for how to think about privacy in OSNs. We contend that the right way to think about privacy is through the lens of *contextual integrity* (CI) [94], which provides concepts that more precisely describe how people conceive of privacy in the real world [91, 92] and therefore should guide how we design OSNs. Rather than focus on control and restriction, CI promotes an overarching idea of privacy as *appropriate flows of information*, the details of which have been applied to environments where privacy settings are well-understood, imperative, and nuanced [14, 36, 81, 95]. We argue that we can begin to apply CI to less codified privacy contexts; in particular, we can apply contextual integrity to OSNs.

In this chapter, we propose Compass, a new social network design that is built with CI as its privacy core. The core idea of Compass is to create a universe of *contexts* where each context is reflective of privacy norms and practices in a specific

real-world context. In Compass, a small collection of users (privacy experts) act as "administrators" to create new context definitions, which are associated with a set of roles and norms. New context definitions that are meant to be public need to be vetted by Compass context creators before being publicly posted; users can create private context definitions in Compass which are not public. In the common use case, Compass allows users to search for appropriate context definitions (publicly defined) and create context instances which they control by assigning roles to users who subscribe to that context instance.

By decoupling the definition of new contexts from most users, Compass aims to significantly simplify the actions required of a normal user of a system. A normal user who joins the system needs to only choose their roles within context instances of which they are a member. In other words, users join contexts for which norms are given – users do not modify information flows. Any information posted by a user is associated with a specific context and the norms of the context govern how the information can be shared within the context. By definition, information flow is restricted within each instance of a context. Users may belong to multiple contexts but, more importantly, information in one context cannot flow to other contexts.

To handle information flow across contexts, Compass supports the idea of "meta-contexts" with norms that bridge two contexts. If an adversary launches a copy-and-paste attack, we use simple similarity matching to detect potential norm violations (or "surprises") for original posters. A surprise notice enables users to signal to Compass regarding the appropriateness of the information flow. Thus, the way that norms are described and used give Compass users well-understood access control behavior as well as recourse when privacy surprises occur.

Compass compiles sets of context-sensitive informational norms (written in propositional logic) to access controls. Compass transforms norms from human-readable logic [14] to a satisfiability problem for which we can (1) efficiently generate the binary decision diagram (BDD) for a norm set, (2) check adherence to norms, and (3) determine how to push a new post to a context in accordance with the norms. We discuss how the BDD formulation is used to check for potential conflicts as norms evolve over time. In addition to framing existing OSNs privacy controls as norms, we present several example contexts to demonstrate how everyday users may make use of Compass' design.

To summarize, Compass offers three key ideas:

1. Decouple privacy rules from users. Contexts and norms are given to users. In Compass, the way that users' posts flow through the system are governed by well-understood sets of rules ("norms") within siloed contexts of individuals. In other words, users do not tune individuals' privacy settings when posting to a context, which is an error-prone process [64, 79]. We have implemented a norm compiler that translates norms into a satisifiability problem data structure, binary decision diagram, and code that evaluate with whom a post should be shared. We describe how privacy rules or norms are written for a context and how they are transformed as inputs to that compiler in Section 4.3. We evaluate the performance of the compiler and its generated code in Section 4.7.

2. As norms of sharing evolve, Compass can detect changes in norms that conflict with prior expectations of privacy. This allows programmers to alert the users to changes from previously established norms of sharing. For instance, one can thus answer the question "Are there individuals who would have access to the data that

Term	Definition
norm	Set of rules governing how information flows between roles within a context.
context	The environment within which privacy must be assessed.
role	Positions within the context that relate users with respect to a context.
transmission principles	The method for transmitting information and the expectations on the nature of the relationship between the transmitter and the receiver(s).

Table 4.1.1: Summary of the important concepts of Contextual Integrity.

did not before the norm change?" (Section 4.3).

3. Auditing and recourse when inappropriate resharing occurs. We implement means to intentionally enable checking when information may have been inappropriately shared in violation of users' expected norms. Original posters can evaluate how to handle the resharing and have multiple recourse options at their disposal. We discuss how our design and extensions to the norm compiler would enable auditing and recourse (Section 4.4).

4.1 Contextual Integrity Review

Nissenbaum proposed CI as a philosophical framework for understanding privacy [94]. In this section, we review CI in order to highlight the relation between Nissenbaum's proposed privacy philosophical framework, Barth, et al.'s work on the formal expression of CI [14], and their potential applicability to the social network environment. Important concepts for understanding CI include contexts, actors, roles, norms, and transmission principles. To illustrate these concepts, we have adapted the work of Barth, et al. [14] to clarify how the conceptual pieces fit together in the framework of CI whose key concepts are summarized in Table 4.1.1. In particular, we draw

from the HIPAA example to guide an understanding of the CI framewor.

4.1.1 Concepts

Contexts are "structured social settings" [94] in which all other concepts operate. Contexts are the social spheres in which the framework of contextual integrity is applied. Thus, contexts may have vague boundaries or membership within the conservative definition of CI. Of course, when establishing context for the purposes of evaluating privacy, one aims for precision in describing all aspects of the context.

Actors are the entities (usually, people or organizations) that are apart of the context between which information flows. Actors are thus the individuals for whom the question of privacy is applied. Each of the actors takes on a *role* within a context. The role is simply a label that enables additional concepts in the context to compute appropriate flows of information. The concept of roles is similar to the concept of human-readable labeling for privacy purposes. However, within Compass, the value of roles is not simply to delineate access orderings [90] but to enable security principles that align with the aim of Compass: least privilege, separation of duties, and data abstraction, as in the case of Role-Based Access Controls [109].

The set of rules that describe how information flows through the context are the context-sensitive informational norms (or, simply, *norms*) of the context. Within CI, norms are not individually-designed rules or "settings" that determine how information should flow. Instead, norms are agreed upon sets of rules that guide how information ought to flow. This is key to understanding the CI perspective of norms: norms are given to the context by its designer. Intuitively, these are the norms that Zuckerberg claimed Facebook changes with respect to privacy online [18]. Through

the perspective of CI, one would say that Zuckerberg argues that the norms of the Facebook context differ from norms of pre-Facebook social interaction.

Norms are given to contexts. These given norms govern how information flows between roles in the context. The CI framework describes how the determination of norms is a function of the actors (transmitter and, optionally specified, receiver(s)), roles of the actors, the subject (information to be transmitted), and the transmission principles.

The *transmission principles* can be thought of as the expectations placed on the recipients of the information. For instance, on Facebook, "friends" (the role that users have with respect to one another) in the default setting post information to Facebook with the expectation (whether intended or not) of symmetry in the way information flows between friends in the network. We can juxtapose this symmetry (or *reciprocity* in Nissenbaum [94]) with the asymmetric sharing patterns of Google+, in which follower relationships make for the ability to post to followers so that followers can read the posters' posts but the original poster may not be able read posts by the followers depending on the circle configurations in the followers' accounts.

Confidentitality is the primary transmission principle that pervades the HIPAA example and, for the purposes of social network privacy, is one of the most common transmission principles in the design of Compass.

4.1.2 HIPAA Privacy Rule Scenario

In Section 5.1 of Barth, et al. [14], the authors discuss how to translate representative examples of the HIPAA privacy rule [96] into a codified programming language

semantics that illustrate the concepts of CI. In the HIPAA example, the context is the hospital in which all of the interplay occurs amongst the concepts described below. The actors in the HIPAA model include the Dr. Alice, Bob, Charlie, and x-ray technician Debbie. With respect to roles, Alice is a doctor, Bob and Charlie are patients, and Debbie in an x-ray technician. The HIPAA examle norms are alluded to by the parenthetical citations and moreover are the rules that govern the interaction between the roles and actors in the context depending on the type of information to be transmitted. We refer the readers to Barth, et al. [14] for the formal grammar description and the example HIPPA policy description.

4.1.3 Rethinking OSN Privacy using CI

To see the shortcomings of existing privacy solutions we use an illustrative example. Consider the primary actor in our example, Alice, a public high school teacher. We also consider the students who attend Alice's school (Stu) and the friends in Alice's social life (Fred). Alice knows that Stu constantly searches for her online account. While Alice does her best to keep sensitive personal information that might lead to her termination (such as a vacation photo) out of the public social network sphere, she must be constantly vigilant of other users' posts that could propagate to people she does not know. In particular, if Fred were to post a photo of Alice to Facebook, depending on *Fred's* permissions, that photo of Alice may be accessible to Stu, which may lead to Alice losing her job.

According to the Facebook for Educators guide [102], Alice could follow the several steps necessary to ensure that information about her online may not propagate to her students. However, users understand poorly how information flows on

Facebook [64] so we have a healthy skepticism about the utility of such guides.

Arguably, framing the fundamental sharing and therefore privacy design of a social network differently would improve Alice the educator's plight. If Fred and Alice shared information within an isolated context in a CI-based OSN, then Alice would not need to worry about her vacation photos "escaping" the context and appearing inadvertently before her students.

The question that Compass must address is how can a user healthily maintain competing social spheres on a single social network? With Compass, our solution revolves around the idea that information sharing should be context-centric with the assumption that the norms of how information propagates in that context are governed by agreed upon norms that have been vetted and applied to the context as a whole.

WHY NOT DIFFERENTIATE? Many methods can be employed to address the issue faced by Alice. Even the use of the Facebook for Educators is one of those. Alice may split her social identity across multiple accounts. Alice may use multiple services to diversify how she broadcasts aspects of her life to individuals and groups she knows in differing contexts. Nevertheless, these tricks complicate the online social experience, encumbering Alice, we argue, unnecessarily.

A social network can be designed to meet the needs of multiple social contexts and roles in a way that more correctly coheres with the expectations of privacy, with the appropriate flows of information, for users like Alice.

4.2 Designing a CI-Aware OSN

In Section 4.1, we reviewed the philosophical framework of contextual integrity. To realize CI within a social network requires mapping CI concepts to concrete structures within a system.

4.2.1 Contexts

Within Compass, a context begins as a user-created entity. Context definition creators have two basic actions for configuring a context: defining roles and choosing a norm set that governs information flows within roles. Compass has two types of context definitions: public contexts and private contexts. Compass promotes a small set of users, who are privacy experts, to articulate context definitions that match with standard privacy expectations with real-world privacy norms. A good example is the HIPAA rule book which articulates privacy rules in the context of healthcare and health records. Individual users can create their own context definitions which are considered private for their own consumption which they manage. Public context definitions are searchable by normal users while private context definitions are not exposed to other users. Private context definitions if more generally applicable can be vetted by privacy experts in Compass before being made public (if the context creator wishes to make it public). To define a public context, a privacy expert user (approved by Compass) articulates a context definition for standard privacy practices in a real-world setting (similar to HIPAA) such as family settings, school, office etc. While certain areas like healthcare might have well-defined privacy policies, similar real-world scenarios (such as an office setting) may promote different context definitions

for the same setting.

A normal user can search for publicly available context definitions and can create an *instance* of a publicly defined context. In the event of multiple context definitions for the same setting, the onus is on the user (creating the context instance) to choose the appropriate context definition for their setting; for example, two different enterprises may choose to adopt two completely different privacy norms for their environment. Any user who creates an instance of a context becomes the "administrator" for the context instance. Private context definitions automatically come with a context instance that the creator manages. An administrator of a context instance has three important actions: adding users, assigning roles to users , and acting on privacy violations (if a user acts in an adversarial manner and is discovered). Compass aims to provide isolation to each context instance; an information posted in one context instance cannot be shared to another context instance. For simplicity of description, due to this isolation principle, we shall use the term *context* to more generally mean an instance of a context definition.

Contexts and norms are pre-configured structures within Compass. By avoiding user-customized settings, we ensure that installed norms can be reasoned about and understood by users and experts who work on the Compass backend. The administrator invites other users to that context and if so begins to apply the other characteristics of the context. Users who join the context are assigned roles by the administrator of the context which defines the flow of information with the context through the norms.

The actions required from a normal user are made extremely simple in Compass: they receive invitations to participate with specific roles in different contexts and

each user can choose to accept or reject such invitations. This is very similar to joining mailing lists or groups. Once they are part of a collection of contexts with specific roles, they can post information, receive posts from other users in the context or repost to users in the context.

CONTEXTS VS SOCIAL NETWORK RELATIONSHIPS: The definition of contexts is completely orthognal to the definition of friends in a social network. Compass operates on top of any existing social network where users have their freedom to make friends and build a social network of friendship relations. The definition of contexts aims to provide better semantic meaning to any friendship link in a social network. If two users have a friendship link, we expect in practice that the two users are part of at least one common context. Any information posted on a social network will be associated with a context and the information flow will be determined by the context and not by the friendship network. In a similar vein, not all users in a context may be direct friends with each other. Hence, if a user receives a post from a friend in a specific context, he/she can forward to other friends within the same context (provided the norms allow it). In addition, any post to a specific context is not automatically shared across all users in the context; information within a context explicitly traverses link by link in the social network. In essence, Compass aims to enforce context-specific privacy norms on top of social network information sharing practices.

Parameter	Definition
transmitter	The actor who is sharing the information.
context	The social sphere in which the post is initially contained.
receiver	The individual(s), role(s), or entire context to which the transmitter to which the transmitter intends to push their post.
attributes	Any additional information that the norms may use to compute (1) to whom the post is pushed, (2) who can search for the post, and (3) who cannot access the post. Compass post attributes are either *context-sensitive* or *context-free*.

Table 4.2.1: Required elements of a post.

POSTS

Compass three actions that users may enact on posts: create, reply, and reshare. A user who creates a post provides the requisite fields (Table 4.2.1) through the Compass interface.

A user who is a member of a context may post information to that context. Depending on the type of the information, the transmitter, and the declared receivers that information may be pushed to certain individuals in the context, searchable by individuals in the context, or entirely inaccessible to other individuals directly through their account in the context.

The user may create a post, indicating who should be the recipient of the information (an individual, role(s), or the entire context) as well as any relevant attributes. The context's norms evaluate how the information should flow.

We have two classes of attributes: *context-free* and *context-sensitive*. Context-free attrbutes belong to publicly defined privacy classes (e.g., contains personally-identifiable information, contains objectionable material). For context-free attributes we have "generic" functions that operate on the message content, regardless of

context.

These generic rules are independent of the context-specific rules and do not affect the BDD construction or performance. A context-sensitive attribute is operates on a post within a specific context.

Compass programmers must translate these attributes to boolean return value functions given the context and the post as arguments.

Replying to a post results in that reply and the original post only being visible according to the norms of the context. Resharing a post outside of a context, however, is governed by the norms of the context as well as inter-context norms (Section 4.5).

NORMS

One set of norms is chosen when creating a context. The set contains multiple norms that decribe how information flows between roles in the context.

How these norms have been created is based on mapping human-readable expectations to propositional logic and codifying the logic in both code that can be evaluated for real-time use by Compass (to evaluate which actors have access to posts) as well as for verification during norm update processes.

We discuss how to translate norms following our presentation of example propositional logic norms is Section 4.3.

In the following subsections, we examine specific examples of OSN usage and the application of CI concepts. The goal is to provide concrete examples that demonstrate how to make the connection between human descriptions of norms to formal logic. This process is very similar to the work by Barth, et al. [14] but focuses on the application to a social network setting. We adopt the Barth, et al. grammar and

notation for expressing the relationships between roles when writing contextually-sensitive norms.

4.2.2 Existing OSN Models in the CI Framework

Facebook

Figure 4.2.1 captures the prevailing Facebook social norms. The simplicity of the norms as they have been written suggest that simplicity does not guarantee flows of information [64]. Given that the role of friend is applied uniformly across all friends of an actor in the Facebook network, the permissiveness of this norm (Norm 4.1) and the Friends of Friends norm (Norm 4.2) are self-evident. That is, the capacity to overshare or experience inappropriate flows of information is high.

Google+

Figure 4.2.2 demonstrates how to articulate the sharing norms on the Google+ OSN. Norm 4.3 illustrates how sharing within a user-generated circle exists. What can be problematic about sharing and privacy on Google+ is what we see in Norm 4.4. In Norm 4.4 we see that a circle member can easily reshare information to users outside of a circle. Google+ does not protect the original posters from the inappropriate flows of information, making the friction to share with users outside of circle context nearly absent. A message indicating that the post was originally a limited share presented to the user, but the extent to which the original poster desires the post to remain within the circle is *not clearly indicated by the context*: no agreed upon norms exist to guide the appropriateness of the flow of information. Consequently, the resharer may inadvertently violate the expectations of the user. This is particularly

problematic given that the original poster has no direct means of accessing with whom the reshared post was shared: the post has escaped the isolation of the context.

4.2.3 Example Contexts in Compass

Apart from the contexts formally described in the Barth, et al. paper (HIPAA privacy rules in the healthcare domain, Children's Online Privacy Protection Act (COPPA), and Gramm-Leach-Bliley Act (GLBA)), we could find very few scenarios where there are publicly available formal norm definitions (or at least sociological descriptions) for specific contexts. Hence, we articulate three specific examples of norms for three different contexts and use these examples in our evaluation. We review the assumptions of the context and examine how one could write the associated logic for the norms.

Family

Many types of family dynamics exist. For the purposes of our examples we focus on a small set of norms to discuss how family members interact in a social network context. We introduce assumptions about norms that we do not assume to be universally applicable; instead, we expect that when the reader will understand the conversion that we apply from the read norms to the propositional logic.

Consider a family structure where family members (actors) have at least one of the following roles: $elder$, $generation$-0 (think: parent), and $generation$-1 (child). Moreover, we have attributes attached to messages that are sent through the context.

We assume that families contain a subset of members who are considered the

mature, wise elders who make decisions about difficult topics. In Norm 4.5, we illustrate the use of the elder role to constrain the flow of information amongst users who have the role of elder when a message contains information about a genetic disease in the family. In the example family context, communication about a genetic disease of a parent is constrained to only the elders of family; e.g., information about Huntington's disease will only be shared with and amongst elders.

Norms can also be topic specific for a generation. For instance, communication about finances remains between parents as in Norm 4.6. However, sharing about the children's low academic performance (Norm 4.7) or throwing parties (Norm 4.8) remains within their generation.

Notably, this example is imperfect. It may not be expressive of the family norms to which some readers are accustomed. The purpose of this subsection has been to demonstrate the means of expressing norms that convey a family dynamic. Different families, cultures, etc. will have different norms.

CLASSROOM

We consider the case of a classroom with students that are divided into teams and an instructor. Students thus have both the role of *student* and as the member of a specific team, which we generically express as *team-member* in Figure 4.2.4. We examine a set of norms that govern the flow of information in various scenarios that affect the classroom dynamic between these roles and the individuals in the classroom.

One norm is an instructor broadcasting announcements to the class. We see this permissive norm as Norm 4.9.

Of course, in the instructor has a specific message pertaining to a specific team in the class, we see that there is a norm to ensure only members of that team receive that message in Norm 4.10. Undoubtedly, team members will want to be able to communicate amongst only themselves too (Norm 4.11).

Questions about one's own grade are only received by the instructor (Norm 4.12). But gossip about a teacher is only seen by students in the class and not the instructor (Norm 4.13).

UNIVERSITY DEPARTMENT

A university department may have a number of nuanced roles: administrators (seniors and aides), professors (tenured, tenure-track, non-tenure-track), students, and staff. On a sensitive matter regarding a students' disciplinary matter, an instructor may send a message that only the administrative board (tenured faculty who hear cases about students) will see (Norm 4.14).

Communication from the chair about the tenure promotional process can only accessible to the tenured members of the department and the administrators (the *faculty-tenure-committee*).

When prefessors are reporting student grades (Norm 4.16) or students reporting course ratings (Norm 4.17) only the administrators of the department see those scores, so as to act as mediators and persistors of initially sensitive information.

4.3 COMPASS PRIVACY NORMS

4.3.1 TRANSLATING LOGIC TO CODE

In order to evaluate human-readable norms as access controls, we implement a norm compiler that translates norms to a satisfiability problem data structure that can be queried regarding access controls. The norms, written as propositional logic, are compiled into *binary decision diagrams (BDD)* with additional generated code to represent a context. Given a post by a user in a context, a query to the BDD returns (1) if the message can be transmitted to users or roles in the context and (2) to which users or roles the message should be sent.

There are two key components to the norm compilation: (1) connecting the grammar symbols to boolean return value functions and (2) generating the BDD.

FROM GRAMMAR SYMBOLS TO FUNCTIONS

From the propositional logic grammar that we adopt from Barth, et al., we must translate each of the propositional logic variables into functions with boolean return values.[1]

inrole() function calls are translated to database queries about the receiver and whether the message is appropriate for their access. When the post is posted, traversal of a p_2 node checks if the explicitly stated receivers meet the inrole() criteria to determine if a message push to their feed is appropriate. Alternatively, if a user requests direct access to a post, traversal of a p_2 node checks if the requestor's

[1] We frame this discussion from the perspective of having implemented a prototype of the norm compiler in C++. Languages with more sophisticated "return" type systems (e.g., Haskell's currying) are future work.

inrole() state is true or false for that node.

$(t \in \ldots)$ functions are translated to attribute (attr()) function calls in our code. These functions exist as either explicit database checks (see if a post has a particular flag set for the attribute) or inferred base on the message content (e.g., vulgarity detection [131]). These functions directly query or operate on the post.

$(q = \ldots)$ variables are translated to subject() function calls. These calls check that the subject of a message is the actor or role specified in the message.

Binary Decision Diagram Review

A Binary Decision Diagram (BDD) is directed, acyclic graph (DAG) representation of a boolean expression. In our example norm sets, all individual norms are represented as conjunctions. To determine whether to accept a post to a context and to whom to post it, we assume that the set of norms is disjunction ("or"-ed together) accurately represents the intentions for information flows.

Benefits of Propositional Logic We ensure the correctness and coherence of the privacy policies in the system is through the use of propositional logic. To demonstrate this translation for the system we use the example norms from Barth, et al. [14]. In Figure 2 of that paper, the authors show how to write the logic corresponding to norms in their linear temporal logic (LTL). Given that propositional logic offers a subset of the facilities of LTL we cannot perfectly codify the norms provided by Barth, et al.[2]

[2]Of course, the LTL that those authors present is itself incompletely expressive. For instance, the authors recognize that the norms are used profilactically rather than *in reaction to* a violation of a norm; that is, the logic does not have the expressiveness to redirect flows of information contingently

This means that all variables required are on the order of $O(2^n)$ in the number of fields that must be tested for a single norm. The contents of a single norm remain small in practice though: in Barth, et al. [14], all norms contained less than 10 boolean expressions, which is a very tractable number of test cases to generate and execute before deployment of any norm.

For the HIPAA Privacy Rule, the functions that create attributes on actors is the inrole function, for which there are the following six types: covered-entity, individual, provider, patient, psychiatrist, and clergy. The attributes of the messages have the following four types: phi, psychotherapy-notes, condition-and-location, and directory-information.

As we have shown with the example norms Figures 4.2.3, 4.2.4, and 4.2.5, in addition to the examples of norms presented in the Barth, et al. paper, the number of boolean variables are within a tractable range for modern computation. As we have seen, $n < 20$ for the $O(2^n)$ complexity for the satisfiability formulation of the norms.

GENERATING THE BDD

Our lexer and parser produce a simple propositional logic AST that maintains the relationship between propositional logic operators (“and”, “or”, “negation”) as well as method call state (function name and arguments).

After the norms are parsed by the compiler, the AST is analyzed to translate every node into a boolean variable, in order to map the function object to a boolean variable. Notably, we keep track of function signatures so as to not overgenerate

based on the failure of another norm.

boolean variables in the BDD analysis.

We use BDDs for representing the problem of satisifying norms within the CI framework. We use a BDD library (in our case, BuDDy [77]) to input variables and propositional logic as a satisfiability problem. More importantly, we use a BDD library to heuristically produce a reordered and more-reduced BDD; in particular, a BDD that has many isomorphisms in its subgraphs merged.

With a DAG representation of the BDD, for a norm set, we maintain a set of vectors of ordered transitons between nodes that represent the satisfiable paths of the DAG. (Each vector in the set represents a satisfiable path in the BDD.) We call this the *satisfiability set*.

Evaluating Context Norms and Posts with BDDs

When considering to whom to push new posts, we assume that the traversal of nodes in a BDD (eventually, to a satisfying node) that contain a receiver (p_2) represent the list of valid receivers for a message. In the case that a receiver is somehow missed due to a short circuiting of a norm (in other words, at least two norms are satisfied by the input, but only one triggers a push), users can currently poll for access to data. Given their state as a receiver, the BDD is traversed querying whether the receiver inrole() function calls match the requestor's profile.

Handling Norm Updates One of the key ideas of Compass is that we provide a framework so that expert programmers create contexts with norms that can be checked for coherence over time. This is enabled by comparing the satisfiability sets of two different norms. Satisfying paths that do not overlap must be checked

against one another to ensure that no bad informational flow surprises have been admitted to the norm set. We say that the system has *extensible verifiability* so that as the contexts evolve – requiring additional roles, nuanced norms, etc. – the framework allows the programmers to make clear what past norms may be violated so as to limit inducing privacy surprises. Any differences, especially in terms of additional satisfying paths, must be inspected to determine if a conflict has significant social impact.

4.4 Surprise Information Flows

The basic design of Compass is designed around the principle of contextual integrity and is not designed to handle adversaries whose explicit goal is to violate the privacy norms. Adversarial users may violate the norms of the original poster by launching a copy-paste attack where they explicitly "copy" information received in one context and post it as new information to users in a different context. Compass explicitly disallows such reposting and adversaries need to perform a copy-paste operation to achieve this goal. In traditional OSNs, an original poster would not know through the OSN that such an attack has occurred.

To detect copy-and-paste (C&P) attacks, we rely on similarity measures for text [16, 86] (we plan to use the library facilities of NLTK [17]), natural images [127], and video [45, 110]. Whenever a new post of a user to a context is deemed very similar to a post received by the same user from a different context, we detect a potential copy-paste attack. We refer to such a case as a *surprise information flow*. If the information appears to be an "exact" copy of the original, then such a post is disallowed and the original poster is alerted when this norm-contrary surprise occurs. If the in-

formation appears to be approximately similar, then the user posting the information is requested to attest the originality of the posting and the original poster is informed of a potential surprise without revealing the new information posted; in this case, the original poster in doubt can request the user posting the new information to either share the posted information or request a similarity report from Compass.

For any surprise information flow, Compass informs the original poster of the retransmitter as well as the context with which the information was shared. When a violation occurs, users may find multiple reasons to take issue with the resharing that occurred. The parameters of a post are extended to include those that the user may find are appropriate to classify the privacy violation: time, new context, new users, the content, etc. Ideally, one would require a reputation system on top of Compass to resolve conflicts across original posters and potential re-posters in the case of multiple conflicting claims about information originality for similar-looking posts. For any declared privacy violation, Compass logs the user's feedback about whether the surprise was acceptable or not and why. The user may then choose to decrement the reputation of the individual who inappropriately reshared the information or the context to which it was reshared. This scoring, relative to the user as well as globally maintained, enables all users to assess the credibility of individuals within the Compass ecosystem. Administrators of contexts have the power to remove users with low reputation (involved in several conflicts) from a context.

Under scenarios where the original poster values sharing the information beyond a context, the original poster has the option of granting permissions to enable surprises. Such reposts can either be treated as new posts (if the origin seeks not to reveal themselves in new contexts) or can be explicitly treated as a repost with a pointer

to the original post. We explicitly aim to constrain reposting across only one level and disallow multiple levels of reposting across contexts. Compass can create a special broad context called "public-posts" which includes all users by default. Posts intended for public consumption can be labeled under the context "public-posts" can be reshared across users.

4.5 Merging Contexts

Up to this point, we have discussed how Compass isolates contexts from one another. Reposts between contexts are considered potential privacy violations (or, neutrally stated, "surprises"). Nevertheless, users may find legitimate reasons to share posts between contexts. The framework of CI acknowledges the existence of metacontexts [94]. Two different university departments may want to encourage collaboration and therefore need to share information about students, research, etc. between roles in each department. Thus, there may be overarching social norms that encourage information to flow between multiple contexts.

To codify metacontexts, programmers would create a new context for which the roles are encapsulated by its subcontexts. More importantly, members of the contexts must agree on a set of norms that describe how information flows between roles in both contexts. In the same way that a set of norms would be chosen for a context, so too with a metacontext.

For instance, the classroom example may have multiple "teaching sections" as separate contexts, for which "assistants" lead the instruction of several "students" from the overarching classroom context. These relationships and the norms that describe how information flows between these contexts also must be described in the

system. In other words, we use the same propositional logic framework to implement sharing across contexts. Recall that posts are created within a context, which means that original posts or reposts are explicitly labeled as being sent through the norms of a specific context (which could be a metacontext).

In order to avoid a hard-to-reason-about transitive norm relationships between chains of contexts (think, Facebook friends of friends sharing settings [64]), posts may not be daisy-chained to disseminate information outside of specified metacontext norms. In other words, given three contexts C_1, C_2, and C_3, if there are norms between C_1 and C_2 and between C_2 and C_3 but none between C_1 and C_2, a message posted to C_1 that flows according to norms to C_2 may not flow to C_3. Until an explicit C_1 and C_3 set of norms has been installed for such a metacontext, posts not sent to such a context do not flow through such a context. In other words, posts only flow in the contexts (including metacontexts) to which they have been explicitly posted.

4.6 IMPLEMENTATION

We have implemented the core functionalities of the Compass design in order to demonstrate concretely how to realise contextual norms of transmission as code and evaluate the complexity of the norms constructs in example contexts. The implementation consists of a compiler that has two features: (1) source-to-source translating norms to access controls with a leveldb [30] database backend and (2) generating binary decision diagrams that describe pathways for evaluating norms to determine whether and how to grant data access.

The code has been implemented in about 1000 lines[3] of C++, flex, and bison as libraries that utilize BuDDy [77] for binary decision diagrams (BDDs) [9], thrift [38] for data structure serialization, leveldb [30] for the key-value stores, as well as bison [31] and flex [72] for the core compiler components. To generate the BDD, the compiler executes two passes over the norms: once to gather the *variables* for the propositional logic and the second to generate the BDD once all variables have been determined.

Once the code is generated, per context, the programmer must complete the logic for the boolean variable functions. For instance, explicitly labeled attributes for a context's posts involve the programmer extending the `post` thrift data structure to include the requisite boolean field and updating the logic of the attribute function call to return the value of the field.

4.7 Evaluation

We have implemented the core functionality of the Compass system. In this section, we evaluate the efficiency of the norm compiler to demonstrate that the feasibility of using Compass' design in on OSN. We evaluate Compass based on the example contexts we described in Section 4.2.3 and formally in Section 4.3. Our evaluation focuses specifically on answering two questions: (a) How well does our compiler perform in generating BDDs for different contexts? (b) What is the efficiency of the verification code generated for verifying information flow for a context based on its defined norms?

[3] SLOCCount (`http://www.dwheeler.com/sloccount/`)

4.7.1 GENERATING BDDS

To address the performance of the compiler, we examine the runtimes of its primary components as it operates on example contexts. In particular, we measure the average runtime of the lexer, parser, and BDD reordering components. We argue that the number of variables from a typical norm set will generate a computationally tractable satisifiabilty. We demonstrate this result through the runtime performance of lexer, parser, and BDD reordering based on our example norms in Figure 4.7.1. The log bar plot illustrates the relative times for each component. As expected, the more complex norm sets require more time for lexing and parsing, but in the sub-100μs range. While the BDD reordering operation can require as much as two orders magnitude more time to compute, the sub-second completion time implies that this operation can be employed for generating new access controls. Our results demonstrates that the time to convert norms into usable code is fast and useful in a OSN environment.

We also present the output of the BDD reordering operation for our example norms in Figures 4.7.2a (Family), 4.7.2b (Classroom), and 4.7.2c (University Department).

4.7.2 GENERATED CODE EFFICIENCY

Given the relative increase in complexity of the access controls with Compass' norm sets versus popular OSN access controls, one question that emerges is how efficient is checking the BDD for an accepting post and determining to whom a post should be written.

In Figure 4.7.3, we show that the number of posts per second that the different generated BDDs can sustain in memory. These trials involved "random" input: for input posts, we randomize attributes based on the possible inputs for the BDD to explore all branches.

More complex norms (Family, University Department, and Classroom) norms can only sustain an order of magnitude less of in-memory queries per second. Nevertheless, we note that the sustain throughput is well-above the system limits that would be expected with networked systems: the number of queries per second is well above the C10K network bottleneck threshold [67].

$$\text{inrole}(p_2, friend) \tag{4.1}$$

$$\text{inrole}(p_2, friend\text{-}of\text{-}friend) \tag{4.2}$$

Figure 4.2.1: Norms for Facebook as a context.

$$\text{inrole}(p_1, circle\text{-}creator) \wedge \text{inrole}(p_2, circle\text{-}member) \tag{4.3}$$

$$\text{inrole}(p_1, circle\text{-}member) \wedge (t \in limited) \tag{4.4}$$

Figure 4.2.2: Norms for a context as defined by a Google+ circle.

$$\text{inrole}(p_1, generation\text{-}0) \wedge \text{inrole}(p_2, elder) \wedge (q = p_1) \wedge (t \in genetic\text{-}disease) \tag{4.5}$$

$$\text{inrole}(p_1, generation\text{-}0) \wedge \text{inrole}(p_2, generation\text{-}0) \wedge (t \in finances) \tag{4.6}$$

$$\text{inrole}(p_1, generation\text{-}1) \wedge \text{inrole}(p_2, generation\text{-}1) \wedge (t \in low\text{-}academic\text{-}performance) \tag{4.7}$$

$$\text{inrole}(p_1, generation\text{-}1) \wedge \text{inrole}(p_2, generation\text{-}1) \wedge (t \in parties) \tag{4.8}$$

Figure 4.2.3: Norms of transmission for a family context.

$$\text{inrole}(p_1, instructor) \wedge \text{inrole}(p_2, student) \wedge (t \in announcement) \tag{4.9}$$

$$\text{inrole}(p_1, instructor) \wedge \text{inrole}(p_2, teamX\text{-}member) \wedge (t \in teamX) \tag{4.10}$$

$$\text{inrole}(p_1, teamX\text{-}member) \wedge \text{inrole}(p_2, teamX\text{-}member) \wedge (t \in teamX) \tag{4.11}$$

$$\text{inrole}(p_1, student) \wedge \text{inrole}(p_2, instructor) \wedge (q = p_1) \wedge (t \in grade) \tag{4.12}$$

$$\text{inrole}(p_1, student) \wedge \text{inrole}(p_2, student) \wedge (t \in instructor) \tag{4.13}$$

Figure 4.2.4: Norms of transmission for a classroom context.

$$\text{inrole}(p_1, instructor) \wedge \text{inrole}(p_2, administrative\text{-}board) \wedge (q = student) \wedge (t \in disciplinary\text{-}matter) \tag{4.14}$$

$$\text{inrole}(p_1, chair) \wedge \text{inrole}(p_2, faculty\text{-}tenure\text{-}committee) \wedge (q = untenured\text{-}faculty) \wedge (t \in tenure\text{-}case) \tag{4.15}$$

$$\text{inrole}(p_1, instructor) \wedge \text{inrole}(p_2, admin) \wedge (q = student\text{-}grade) \wedge (t \in grades) \tag{4.16}$$

$$\text{inrole}(p_1, student) \wedge \text{inrole}(p_2, admin) \wedge (q = instructor) \wedge (t \in course\text{-}rating) \tag{4.17}$$

Figure 4.2.5: Norms of transmission for a university department context.

```
const char *department_norms =
    "(inrole(p1,instructor) && inrole(p2,adboard) && subject(student) && attr(msg, discip)) ||"
    "(inrole(p1,chair) && inrole(p2,factencom) && subject(untenfac) && attr(msg, tencase)) ||"
    "(inrole(p1,instructor) && inrole(p2,admin) && subject(student) && attr(msg,grades)) ||"
    "(inrole(p1,student) && inrole(p2,admin) && subject(instructor) && attr(msg,courserating))";
```

Figure 4.3.1: Example of the norms translated as strings by a programmer for input to norm compiler. Notably, the $(q = \ldots)$ and the $(t \in \ldots)$ variables in the propositional logic of Figure 4.2.5 have been translated to `subject()` and `attr()` function calls. Depending on the context, the `attr()` are customized to either execute explicit field checks in a database or infer whether an attribute is present based on the content of the post.

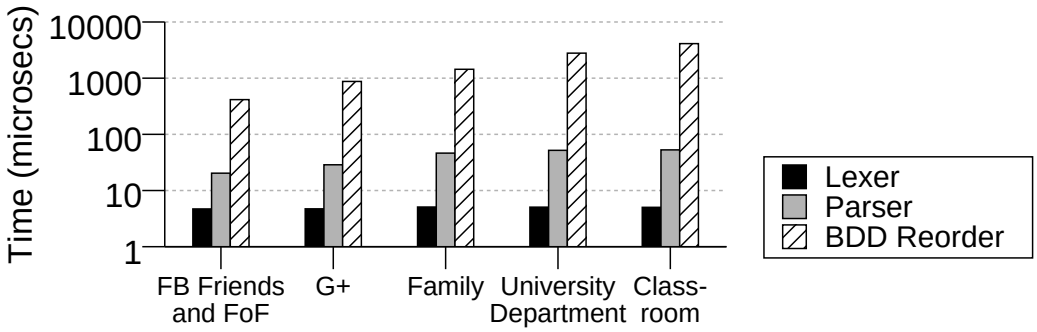
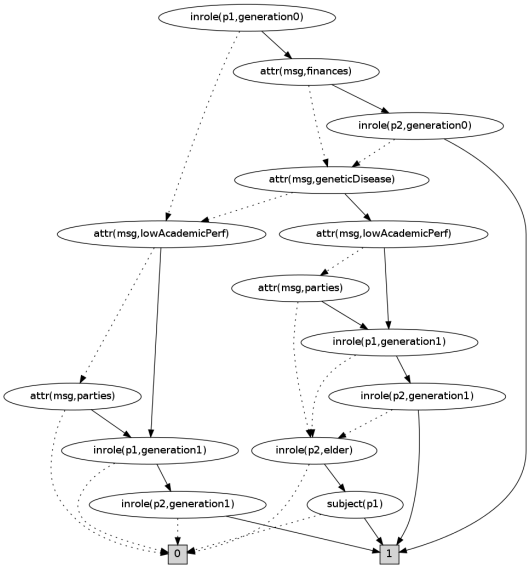
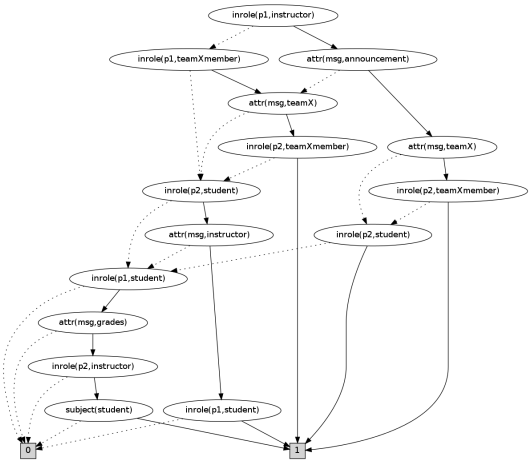


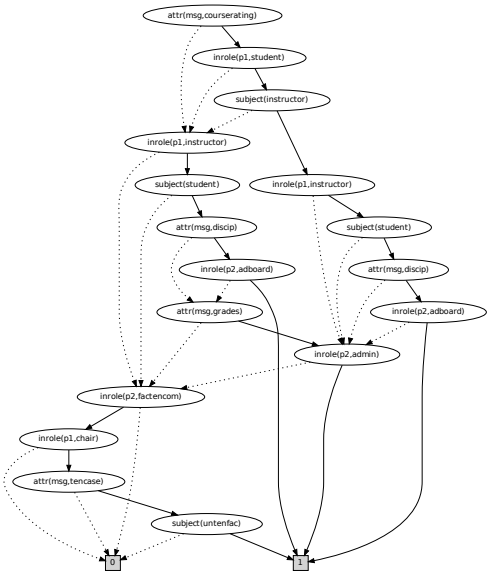
Figure 4.7.1: Compiler Performance. In this log bar plot, we show the average times for different stages of the norm compilation process compared to the norms that we compiled. While the more complicated norm sets require an order magnitude more time, especially for the BDD reordering (we used BuDDy's BDD_REORDER_SIFTITE method which is the most thorough heuristic but also the slowest), the times are quite reasonable given that norm regeneration and installation ought to be a rare process: privacy policies do not change at a subsecond granularity.



(a) The BDD DAG generated by BuDDy that corresponds to the Norms in Figure 4.2.3.



(b) The BDD DAG generated by BuDDy that corresponds to the Norms in Figure 4.2.4.



(c) The BDD DAG generated by BuDDy that corresponds to the Norms in Figure 4.2.5.

Figure 4.7.2: Generated, reordered BDDs for our example contexts.

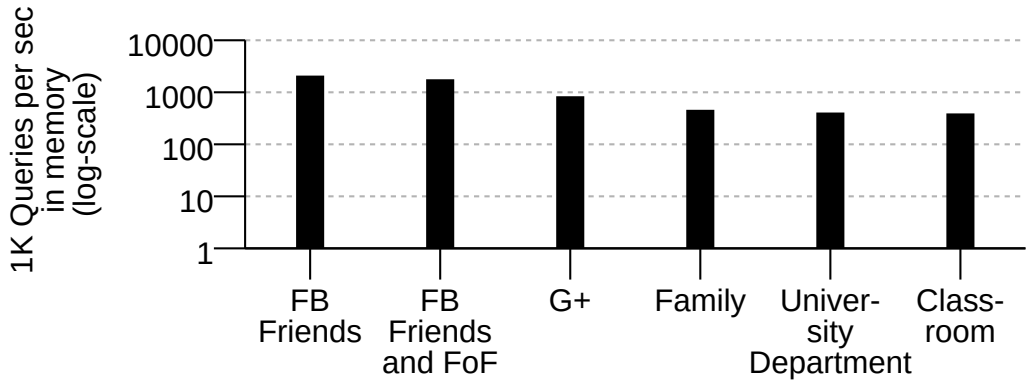


Figure 4.7.3: Context norm performance. In this log bar plot, we illustrate the number of new posts that have their access control flows evaluated with the generate Compass code. These measurements involve the traversal of the corresponding norm BDDs. While more complex norm sets incur a performance penalty – for instance, note that for "FB friends" norms we sustain four times as many queries per second – the naive Compass code generation sustains queries per second well above the C10K queries per second network bottleneck, which is at 10,000 queries per second.

5

Related Work

5.1 Private Photo-Sharing

P3 [106] examined the use of non-colluding services to store minimally-revealing cleartext images in one service and encrypted versions of DCT coefficients of JPEG images in another service. Their system experienced a 10-20% file size increase from the original compressed image when one follows their recommended privacy-preserving settings by setting the DCT-hiding threshold in the range $T \in [10, 20]$. The authors acknowledged their technique's vulnerability to face identification when $T \geq 35$. Cryptagram fundamentally differs from P3 in two ways. First, Cryptagram completely avoids the use of third parties. Secondly, Cryptagram works only in

the encrypted bit space and does not expose any unencrypted data to the end-user. Unless users' keys are compromised, users cannot have their faces detected with any of our embedding protocols.

Steganography. Cryptagram is superficially reminiscent of various attempts to embed cryptographic data in JPEG through traditional steganographic techniques [48], but differs significantly from conventional JPEG steganography. Cryptagram makes obvious that it is hiding data to attain greater efficiency, and furthermore, does so in a way that is robust to image compression, which steganography generally is not. Attempts to achieve lossy compression tolerant steganography are early works with inefficient results [62].

One recent work attempted to embed data in JPEG DCT coefficients without the steganographic constraint of being hidden. The non-linearities of DCT quantization and rounding in standard compression and decompression required very conservative data embedding that resulted in efficiency significantly lower than what we were able to achieve [11].

Li et al. [75] address the concern of hiding data within DCT coefficients but shuffle the DCT coefficients between blocks that then are quantized during the JPEG compression algorithm. Likewise, Poller et al. demonstrate that DCT coefficient permutation and spatial domain permutation do provide some security features but do not address efficiency or prove the correctness of their security mechanism [103].

A formalization for the description of the embeddings that we use in Cryptagram have been explored by Galand and Kabatiansky [49] but the authors do not explore how to construct such protocols.

But Cheddad et al. [23] claim that spatial domain techniques are not robust against

System	File Manager Integration	Versioning	Client-Side Encryption	Server Equivocation	Provider Independence	License
Dropbox [5]	✓	✓	✗	✗	✗	✗
TrueCrypt [10] + Dropbox	✓	✓	✓	✗	✗	✓
Jungle Disk [1]	✗	✓	✓	✗	✗	✗
SparkleShare [2]	✓	✓	✗	✗	✓	✓
SpiderOak [3]	✗	✓	✓	✗	✗	✗
Syncany [4]	✓	✓	✓	✗	✓	✓
Ubuntu One [22]	✓	✗	✗	✗	✗	✓
Wuala [71]	✗	✓	✓	✗	✗	✗
SPORC [42]	✗	✓	?	✓	?	✓
Lockbox	✓	✓	✓	✓	✓	✓

Table 5.2.1: Competing projects and features. On the License category, ✓ indicates open source, ✓ a mixture of open source and proprietary software, and ✗ proprietary software.

noise, only work in bitmap-formatted images, and are not robust against lossy compression and image filters. Cryptagram overcomes all of these drawbacks in spatial domain embedding and demonstrates the useful privacy and security properties that can be available for OSN photo sharing. Cryptagram achieves q, p-Recoverability in the face of the complete recompression of the JPEG image containing sensitive information.

5.2 Lightweight Private File-Sharing

Lockbox aims to achieve several goals that either independently or in strict subsets have been pursued by related projects. We examine these ideas below as well as how they relate or have influenced the design of Lockbox.

Lockbox seeks to resolve a tension in the landscape of the values design space that has been discussed by Stark and Tierney [116]. That Lockbox discussion outlined

the necessity for privacy for *user empowerment* [25] to be held with at least equal importance to convenience and usability in the design of storage solutions that leverage the cloud.

Delta Encoding: The idea of applying rolling checksums for the purposes of delta encoding has been widely deployed in the utility `rsync` [124]. Algorithmically, these ideas relate to Rabin fingerprinting [107].

The Low-Bandwidth File System [89] has demonstrated the utility of rolling checksums and caching on a file system level. This complicates the matter of making it easy for users to adopt our utility. While algorithmically sufficient, requiring users to configure a new file system arguably defeats the aim of usability and wide adoption.

Several projects have examined how to reduce web traffic bandwidth consumption by eliminating redundant transmissions [60, 115]; however, delta encoding and "differencing" techniques that were applied make sense on unencrypted network data and are therefore orthogonal to our application assumptions.

Deduplication with Encrypted Data: Convergent Encryption [32] is an idea that can be leveraged in a similar scenario as ours. Any file to be encrypted uses a hash of its file contents as the key. In this way, only individuals with access to the cleartext file contents can infer the key without brute-forcing a solution. Of course, this is also the drawback with the approach is that any duplicated file can be used to confirm the corresponding encrypted contents. POST [88] demonstrates the use of convergent encryption in a storage environment where the poster remains unaffiliated with the posted content. Thus, while an attacker who guesses the cleartext of a message can verify its existence on the posting service, the original poster evades detection. In

DOT, the authors argued for the use of convergent encryption in their system to enable caching and retrieval [123].

(Dis)Trusting the Cloud: Oceanstore [70] showed the design for a secure, distributed file system. Pond [108] demonstrated an early prototype of the system. Use of the cloud simplifies many of our design considerations. We believe that these techniques are still valid and perhaps influenced the internal design of cloud service providers on which we tested our system.

Bayou [121] presents an evaluation of conflict resolution that occurs at the mobile device user level. In our system, we actually leverage the consistency models of existing web service providers (S3 has atomic writes but eventually consistent reads of raw data while SimpleDB has the option to enable consistent reads of metadata.)

Distrusting the cloud has been a common thread of research for several years. Cachin, et al. [21] review the methods by which one could come to trust the cloud. Mazières, et al. [84, 85] discuss the design considerations behind the realization of a file system that does not have to trust the storage service. Depot [82] summarizes and introduces the various consistency models. SPORC [42] introduced the question: What if we don't trust the cloud to order our messages correctly? Starting with this question, overhead enters the picture. A point of comparison in our system is whether such a design consideration presents an unnecessary overhead in designing a cloud-based user application. SUNDR [73] introduced an earlier notion of SPORC's fork*-consistency with fork-consistency in the case that malicious remote file systems forked or presented divergent views of the storage to different users.

Encryption on top of Dropbox: TrueCrypt [10] enables users to symmetrically encrypt volumes (or directories) on local filesystems. To achieve a form of encrypted

file sharing one can then encrypt the Dropbox directories with TrueCrypt. This provides the expected guarantees of block ciphered volumes but conveniently wrapped to provide transparent encryption for end-users.

The challenge of course is then how to share the files. While Dropbox enables the encrypted files to be delivered to other users – Dropbox solely facilitates the sharing process – the users must then also properly setup TrueCrypt to use the same keys. The consequence of using the same symmetric keys however is that any granularity is sharing becomes difficult to manage. Given that TrueCrypt is meant to operate over a directory and its subdirectories, any nuanced permissions are unable to be enabled with this system.

5.3 Contextual Integrity and Online Social Network Design

New systems have emerged to enable users to think differently about their privacy in OSNs [15, 98]. Several projects influence, compare, and contrast with the design of Compass. Recently, Barkhuus examined the application of CI to the considerations of privacy in HCI works [13]. However, the most closely works is Aegis [66], which takes a semantic web approach to writing policies for social networks. While the authors claim to have implemented contextual integrity, their example contexts and policies reflect basic access controls (user, requested resource, and read/write access) akin to UNIX access controls, which ignore the underlying principles of contextual integrity (namely, norms with roles and attributes are completely ignored and therefore express extremely limited forms of norms and contexts are presented in Aegis).

Compass is a step in a new direction as it connects formal methods approaches to privacy [63] with a tangible implementation.

Social circles [8] was the first proposal for an interface for visualizing clustering information computed from end-users' relationships [41].

Lipford, et al. [78] argue from the perspective of contextual integrity for how social networks ought to be designed for users' privacy. The authors emphasize user awareness in the UI of the system, using prototype implementations on top of existing social networks to suggest how users can be made more aware of how the information they share propagates in the social network. This is strictly limited to the one-hop "friends" dynamic of Facebook.

Guha, et al. [57] argue for a system that shares information in plainsight where users share data through publicly known dictionaries and privately shared seeds for the substitution cipher in the system. The authors argue that the scattering known data, making it harder to combine data for reconstruction, embodies aspects of contextual integrity to ensure the appropriate flow of information.

Koi [58] demonstrates how to verify the design of a location-privacy platform for mobile device apps. While Compass can incorporate the use of a verification platform into its design during the creation of new contexts and policies, we have focused on simple API designs that better match users' expectations for describing information flows.

Anwar, et al. [47] illustrates how to articulate the Facebook access controls in an access control model. Fong then described the use of such models to articulate how to mitigate Sybil attacks against a Facebook social network [46].

Fang and LeFevre developed the notion of the "privacy wizard", which takes

preferences from users in order to construct the appropriate privacy settings for the user [41], which is an approach that goes against the grain of Compass. Compass emphasizes the use of understandable, given norms and roles within a context that enables explicit reasoning about the flow of information.

Labeling paradigms with data confidentiality guarantees offer a different model to privacy in social networks [12].

Systems solutions have been presented and suggest methods of applying existing labeling theories to new domains to achieve security [52, 69, 126, 132, 133].

6

Conclusion

6.1 Summary of Contributions

6.1.1 Cryptagram

The advent of popular online social networking has resulted in the compromise of traditional notions of privacy, especially in visual media. In order to facilitate convenient and principled protection of photo privacy online, we have presented the design, implementation, and evaluation of Cryptagram, a system that efficiently and correctly protects users photo privacy across popular OSNs. We have introduced q, p-Recoverability and demonstrated Cryptagram's ability to embed cryptographic primitives correctly to attain q, p-Recoverability through JPEG compression in our

implementation.

6.1.2 LOCKBOX

We have presented the design, implementation, and evaluation of Lockbox, a system that unites several technologies to provide users with a usable lightweight file sharing service that affords minimal cost versioning as well as privacy guarantees, even in the face of relevant threats in the era of cloud data storage. This thesis has demonstrated the viability of such a system for enabling privacy for everyday users while maintaining a pricipled foundation for security guarantees.

6.1.3 COMPASS

We introduce a design for a social network that offers a unique set of characteristics that we believe the systems community would appreciate as well as debate. The sharing capabilities within Compass promotes an understanding of and adherence to other users' norms by assigning reputation values to the way in which users encourage the dissemination of other users' data. How we track and ensure the correctness of flows and calculations of the aforementioned scores occurs within the implemented formal semantics of the contextual integrity privacy framework. By focusing on enabling a correct underlying mechanism for promoting the sharing of data while balancing users' intentions, we believe that we have presented a design that most fully embodies and empowers user privacy in an online social network.

6.2 Future Work

6.2.1 Cryptagram

The initial user study for Cryptagram has proved promising: users have been inspired to use this product for photo privacy in OSNs and beyond. Realizing that end-to-end privacy is possible with the modern web, we intend to evaluate additionally requested features, more efficient protocols, and integration with the photo sharing experience that users expect. Usability and privacy are not diametric opposites and Cryptagram aims to demonstrate that the two can be reconciled while maintaining efficiency.

6.2.2 Lockbox

As part of the initial vision of Lockbox, we intend to see our product used by everyday users. Anecdotally and academically, we have witnessed the ever burgeoning demand for secure file sharing while counterbalancing the need to trust remote servers and employing affordable services. Lockbox's initial design points to a promising future that balances usability, affordability, and privacy for everyday users.

Object Store Offload to the Cloud: We note that the use of the object store in the current design enables fast recovery of old versions of files since the data is locally available. Nevertheless, we acknowledge that there might be value in the offloading of non-recent versions to the cloud. We aim to ask the question how should we balance the storage of some data locally and other data in the cloud? What are the criteria for the storage choices that we make in the system?

EVALUATING SNAPSHOT/DELTA TRADEOFFS: When a Lockbox client spends more time and network bandwidth fetching deltas than would be required to fetch a snapshot, an opportunity for optimization is apparent in the Lockbox design. As an optimization, if there are too many deltas, the design enables "garbage collection" of the deltas (compression of the deltas according to file access permissions) with computation executed on a user's client. Determining relevant heuristics that allow clients to decide the correct time to snapshot is important. Currently, snapshotting is guaranteed to occur when a user joins or leaves a shared directory group.

ADAPTIVE DELTA ENCODING: Given the fine grain modifications of Lockbox enables versioning, the system can leverage known techniques in format changes to reconcile the changes that we see in the presented documents. In other words, we may be able to automatically merge conflicts depending on file types. For instance, a change to plaintext documents could be reconciled manually based on known diff techniques while for binary files we could use Courgette [56].

ADVANCED CONFLICT RESOLUTION: Recent work has focused on the use of Operational Transformations (OT) [37] to encapsulate the modifications that can be expected on a text document. In the context of ensuring integrity with the model of fork consistency, SPORC [42] demonstrated how to use chained keys to ensure that OT transactions could be verified in a cloud text editor environment to detect malicious database administrators. One can conceivably ensure that all changes to these types of documents can be leveraged to show that the hybrid cryptosystem primitives can be leveraged to ensure that the OT documents are encrypted and

preserved, without having to encrypt all of the document. The integration of OT with the cloud as the OT conduit is an area of future privacy research.

6.2.3 COMPASS

HOW TO GENERALIZE THE APPLICABILITY OF CONTEXTS? One of the pragmatic concerns for Compass is how to make the various contexts (and their entailed properties) understandable by people? With all of the variety exists in the social dimensions of online social life, how can the users know what they are getting themselves into with a choice of norms, roles, etc.? This is one of the usability challenges that we propose.

GENERATING MECHANISMS FROM POLICIES. Hails [52] examined the utility in presenting policies for information flow enforcement as the same language as the implementation of that policy. This simplification for information flow enforcement is important: the process of translating more human-readable policies into the actual enforcement mechanism can be error-prone if the engineering process is a human-written translation.

Machine checking (as presented in Section 4.6) and ideally machine generation of the norm policies will result in fewer privacy surprises at runtime.

INFERRING NORMS. In the vein of connecting machine learning and privacy [83], we expect that initially written norms will be "buggy" in that they inaccurately capture users' true expectations of privacy. As privacy surprises are logged, we intend to create a system that learns from these surprises what the norms *ought to be* for a context.

Appendix A

JPEG Review

JPEG is a lossy codec designed to provide reasonable tradeoffs between compression and recoverability of the original image [125]. We chose JPEG because of its prevalence on the web and the availability of efficient JPEG libraries. Here we review the core elements of JPEG that affect our protocol design.

A.1 Encoding

Transformations The first step in compressing an input bitmap image, M to JPEG is a color space transformation. In particular, JPEG transforms every pixel in M from the RGB to the YC_bC_r color space through a linear transformation that converts red, green, and blue (RGB) pixel values to luminance (Y), chrominance

blue (C_b), and chrominance red (C_r) values.

$$
\begin{aligned}
Y' &= (0.299 \cdot R'_D) + (0.587 \cdot G'_D) + (0.114 \cdot B'_D) \\
C_B &= 128 - (0.168736 \cdot R'_D) - (0.331264 \cdot G'_D) + (0.5 \cdot B'_D) \\
C_R &= 128 + (0.5 \cdot R'_D) - (0.418688 \cdot G'_D) - (0.081312 \cdot B'_D)
\end{aligned}
$$

After the color space transformation, the JPEG algorithm transforms the three color channels of YC_bC_r independently.

SUBSAMPLING Following the initial color space transformation, JPEG subsamples the color channels. In the default `libjpeg` codec settings, luminance is not subsampled while chrominance blue and chrominance red data undergo 2:1 subsampling vertically and horizontally, also known as "4:2:0" subsampling. This results in one-fourth the number of pixels that represent each of the original chrominance blue and chrominance red channels.

DISCRETE COSINE TRANSFORM The output matrices of subsampling are then transformed using the Discrete Cosine Transform. Since the DCT for JPEG operates only on 8×8 pixel blocks, JPEG breaks each of the subsampled spaces into non-overlapping 8×8 pixel blocks. We refer to these units as JPEG Pixel Blocks or JPBs.

The DCT is a well-understood transformation from the spatial to frequency domain. We present the two dimensional DCT here:

$$G_{u,v} = \sum_{x=0}^{7} \sum_{y=0}^{7} \alpha(u)\alpha(v) g_{x,y} \cos\left[\frac{\pi}{8}\left(x + \frac{1}{2}\right)u\right] \cos\left[\frac{\pi}{8}\left(y + \frac{1}{2}\right)v\right],$$

where $u \in \{0, 1, \ldots, 7\}$ is the horizontal spatial frequency; $v \in \{0, 1, \ldots, 7\}$ is the vertical spatial frequency;

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } u = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$$

is a normalizing scale factor to maintain orthonormality; $g_{x,y}$ is the pixel value at coordinates (x, y); and $G_{u,v}$ is the DCT coefficient at coordinates (u, v). The two

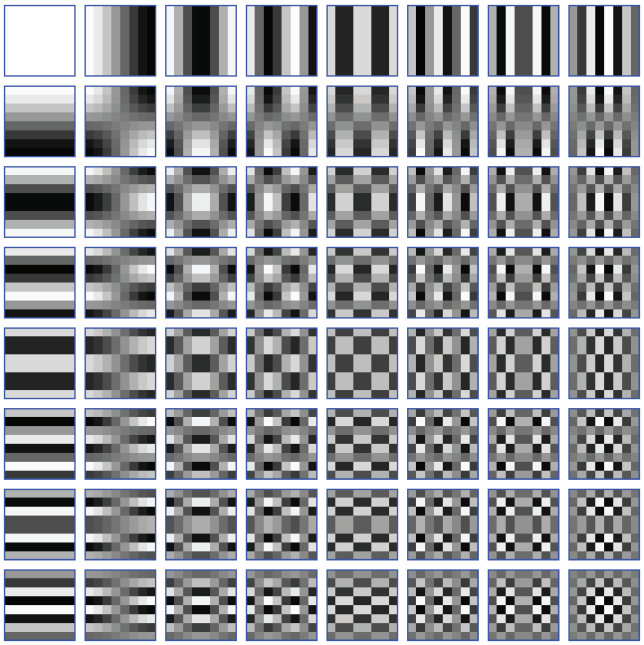


Figure A.1.1: Visualization of the 64 DCT basis functions.

dimensional DCT, computed on each 8×8 block of pixels in an image, results in

64 coefficients per block; the visual representation of which is in Figure A.1.1.

QUANTIZATION The DCT returns real values but we have to represent them on disk with limited precision. The JPEG codec stores each of the 64 coefficients not as a float but as a single 8-bit number that, combined with the quantization matrix, can approximate a real number.

Here is the standard luminance quantization table in JPEG.

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

And for chrominance:

$$\begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

This means, for example, that to approximate the value 99.0 for the 0^{th} (pure DC) component, we would write the integer $\lfloor \frac{99.0}{16} \rfloor = 6$. In short, the lower the value in the quantization matrix, the greater the precision for preservation of that particular DCT coefficient. By scaling all values in the quantization table, JPEG prioritizes lower frequencies (which are more obvious to the human eye) while allowing end-users to achieve a range of qualities.

Chrominance red and chrominance blue use a separate quantization matrix which is more heavily quantized, since the human eye is less sensitive to variations in chrominance than luminance [125].

ENTROPY CODING Once JPEG computes the quantized values, JPEG losslessly writes these values to disk. This deterministic operation provides the biggest savings for bytes on disk. In particular, values are zigzag encoded, meaning that the order of values written to disk follows a zigzag pattern. Any repeated values in the zigzag ordered list JPEG writes as a compressed value; e.g., the JPEG algorithm

dictates writing 20 sequential 0's as $20\{0\}$. The benefits of this process is a reduced amount of disk space required to represent the values.

A.2 DECODING

DEQUANTIZATION Given the original JPEG quantization matrix in the JPEG file, the decompression algorithm multiplies the values read from disk with the corresponding quantization matrix entries.

INVERSE DCT JPEG applies the inverse DCT function to each set of 64 dequantized coefficients to produce the lossy output 8×8 block of pixels.

SHIFTING AND UPSAMPLING CHROMINANCE Values in all of color spaces are then shifted up by 128 to be within the valid display ranges again. JPEG upsamples chrominance accordingly.

CONVERSION FROM YC_bC_r TO RGB Finally, JPEG executes the last linear transformation between YC_bC_r to RGB to present a human comprehensible RGB image.

$$R = Y + 1.402 \cdot (C_R - 128)$$
$$G = Y - 0.34414 \cdot (C_B - 128) - 0.71414 \cdot (C_R - 128)$$
$$B = Y + 1.772 \cdot (C_B - 128)$$

Appendix B

Creative Commons License

This is the text of Creative Commons Attribution-NonCommercial-NoDerivs License, version 3.0.[1]

B.1 LICENSE

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

[1] See http://creativecommons.org/licenses/by-nc-nd/3.0/us/.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. **Definitions**

   (a) **"Collective Work"** means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with one or more other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.

   (b) **"Derivative Work"** means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.

(c) **"Licensor"** means the individual, individuals, entity or entities that offers the Work under the terms of this License.

(d) **"Original Author"** means the individual, individuals, entity or entities who created the Work.

(e) **"Work"** means the copyrightable work of authorship offered under the terms of this License.

(f) **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

2. **Fair Use Rights.** Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. **License Grant.** Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   (a) to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works; and,

   (b) to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Derivative Works. All rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Sections 4(d) and 4(e).

4. **Restrictions.** The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

   (a) You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of a recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. When You distribute, publicly display, publicly perform, or publicly digitally perform the Work, You may not impose any technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies

to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by Section 4(c), as requested.

(b) You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

(c) If You distribute, publicly display, publicly perform, or publicly digitally perform the Work (as defined in Section 1 above) or Collective Works (as defined in Section 1 above), You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution (“Attribution Parties”) in Licensor’s copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; to the extent reasonably practicable, the

Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collective Work, at a minimum such credit will appear, if a credit for all contributing authors of the Collective Work appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this clause for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

(d) For the avoidance of doubt, where the Work is a musical composition:

i. **Performance Royalties Under Blanket Licenses.** Licensor reserves the exclusive right to collect whether individually or, in the event that Licensor is a member of a performance rights society (e.g. ASCAP, BMI, SESAC), via that society, royalties for the public performance or public digital performance (e.g. webcast) of the Work if that performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

ii. **Mechanical Rights and Statutory Royalties.** Licensor reserves

the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions), if Your distribution of such cover version is primarily intended for or directed toward commercial advantage or private monetary compensation.

iii. **Webcasting Rights and Statutory Royalties.** For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

5. **Representations, Warranties and Disclaimer**

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND ONLY TO THE EXTENT OF ANY RIGHTS HELD IN THE LICENSED WORK BY THE LICENSOR. THE LICENSOR MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITH-

OUT LIMITATION, WARRANTIES OF TITLE, MARKETABILITY, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. **Limitation on Liability.** EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. **Termination**

   (a) This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collective Works (as defined in Section 1 above) from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   (b) Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwith-

standing the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   (a) Each time You distribute or publicly digitally perform the Work (as defined in Section 1 above) or a Collective Work (as defined in Section 1 above), the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   (b) If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

   (c) No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

   (d) This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here.

Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

B.2 Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identied itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at http://creativecommons.org/.

References

[1] Jungle Disk. https://www.jungledisk.com/.

[2] sparkleshare. http://sparkleshare.org/.

[3] SpiderOak. https://spideroak.com/.

[4] Syncany. http://www.syncany.org/.

[5] Dropbox: The Inside Story Of Tech's Hottest Startup, May 18 2011. URL http://www.forbes.com/sites/victoriabarret/2011/10/18/dropbox-the-inside-story-of-techs-hottest-startup/.

[6] UNITED STATES v. JONES, 2012.

[7] watchdog 0.6.0: Filesystem events monitoring, 2012. URL http://pypi.python.org/pypi/watchdog.

[8] Fabeah Adu-Oppong, Casey K Gardiner, Apu Kapadia, and Patrick P Tsang. Social circles: Tackling privacy in social networks. In *Symposium on Usable Privacy and Security (SOUPS)*, 2008.

[9] Sheldon B. Akers. Binary decision diagrams. *Computers, IEEE Transactions on*, 100(6):509–516, 1978.

[10] TrueCrypt Developers Association et al. Truecrypt-free open-source disk encryption software for windows 7/vista/xp, mac os x, and linux, 2009.

[11] Julian Backes, Michael Backes, Markus Dürmuth, Sebastian Gerling, and Stefan Lorenz. X-pire! - a digital expiration date for images in social networks. *CoRR*, abs/1112.2649, 2011.

[12] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: an online social network with user-defined privacy. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 135–146. ACM, 2009.

[13] Louise Barkhuus. The mismeasurement of privacy: using contextual integrity to reconsider privacy in hci. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, pages 367–376. ACM, 2012.

[14] A Barth, A Datta, J C Mitchell, and H Nissenbaum. Privacy and contextual integrity: framework and applications. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*.

[15] Ames Bielenberg, Lara Helm, Anthony Gentilucci, Dan Stefanescu, and Honggang Zhang. The Growth of Diaspora - A Decentralized Online Social Network in the Wild. In *IEEE INFOCOM 2012 - IEEE Conference on Computer Communications Workshops*.

[16] Mikhail Bilenko and Raymond J Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48. ACM, 2003.

[17] Steven Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics, 2006.

[18] Bianca Bosker. Facebook's Zuckerberg Says Privacy No Longer A 'Social Norm', Jan 2010. http://www.huffingtonpost.com/2010/01/11/facebooks-zuckerberg-the_n_417969.html.

[19] danah m. boyd and Nicole B. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1):210–230, 2007. ISSN 1083-6101. doi: 10.1111/j.1083-6101.2007.00393.x. URL http://dx.doi.org/10.1111/j.1083-6101.2007.00393.x.

[20] Peter J Burt. Fast filter transform for image processing. *Computer graphics and image processing*, 16(1):20–51, 1981.

[21] C. Cachin, I. Keidar, and A. Shraer. Trusting the cloud. *ACM SIGACT News*, 40(2):81–86, 2009. ISSN 0163-5700.

[22] Canonical Ltd. Ubuntu One. https://one.ubuntu.com/.

[23] A. Cheddad, J. Condell, K. Curran, and P. Mc Kevitt. Digital image steganography: Survey and analysis of current methods. *Signal Processing*, 90(3): 727–752, 2010.

[24] Silent Circle. Silent Circle — Global Encrypted Communications Service, 2013. URL https://silentcircle.com/.

[25] David D Clark, John Wroclawski, Karen R Sollins, and Robert Braden. Tussle in cyberspace: defining tomorrow's internet. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 347–356. ACM, 2002.

[26] Russell B Clayton, Alexander Nagurney, and Jessica R Smith. Cheating, breakup, and divorce: Is facebook use to blame? *Cyberpsychology, Behavior, and Social Networking*, 2013.

[27] Ron Crandall. Some notes on steganography. *Posted on steganography mailing list*, 1998.

[28] James L Crowley. A representation for visual information. Technical Report CMU-RI-TR-82-07, Robotics Institute, Pittsburgh, PA, November 1981.

[29] Al Daniel. CLOC: Count Lines of Code. URL http://cloc.sourceforge.net/.

[30] Jeff Dean and Sanjay Ghemawat. LevelDB. https://code.google.com/p/leveldb.

[31] Charles Donnelly and Richard M Stallman. *Bison: The YACC-compatible Parser Generator (November 1995, Bison Version 1.25)*. Free Software Foundation, 1998.

[32] J.R. Douceur, A. Adya, W.J. Bolosky, P. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 617 – 624, 2002. doi: 10.1109/ICDCS.2002.1022312.

[33] David Duce and T Boutell. Portable network graphics (png) specification. *Information technology ISO/IEC*, 15948:2003, 2003.

[34] Mark Duell. Mark Zuckerberg's private Facebook photos revealed: Security 'glitch' allows web expert to access billionaire's personal pictures. *The Daily Mail (MailOnline)*, December 2011. URL http://www.dailymail.co.uk/news/article-2070749/Facebook-security-glitch-reveals-Mark-Zuckerbergs-private-photos.html.

[35] Mark Duell. Mark Zuckerberg's private Facebook photos revealed: Security 'glitch' allows web expert to access billionaire's personal pictures. *The Daily Mail (MailOnline)*, December 2011. URL http://www.dailymail.co.uk/news/article-2070749/Facebook-security-glitch-reveals-Mark-Zuckerbergs-private-photos.html.

[36] The Economist. The logic of privacy: A new way to think about computing and personal information, 2007. URL http://www.economist.com/node/8486072.

[37] C.A. Ellis and S.J. Gibbs. Concurrency control in groupware systems. *ACM SIGMOD Record*, 18(2):399–407, 1989. ISSN 0163-5808.

[38] Facebook. Apache Thrift. http://thrift.apache.org.

[39] Facebook. Tornado web server, 2009. URL http://developers.facebook.com/news.php?blog=1&story=301.

[40] Hanni M. Fakhoury. Technology and Privacy Can Co-Exist. URL http://www.nytimes.com/roomfordebate/2012/12/11/privacy-and-the-apps-you-download/privacy-and-technology-can-and-should-co-exist.

[41] Lujun Fang and Kristen LeFevre. Privacy wizards for social networking sites. In *Proceedings of the 19th international conference on World wide web*, pages 351–360. ACM, 2010.

[42] A.J. Feldman, W.P. Zeller, M.J. Freedman, and E.W. Felten. SPORC: Group collaboration using untrusted cloud resources. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, page 1. USENIX Association, 2010.

[43] Ariel J Feldman, Aaron Blankstein, Michael J Freedman, and Edward W Felten. Social networking with frientegrity: privacy and integrity with an untrusted provider. In *Proceedings of the 21st USENIX conference on Security symposium, Security*, volume 12, 2012.

[44] Klaus Finkenzeller. *Data Integrity*. Wiley Online Library, 2003.

[45] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, et al. Query by image and video content: The qbic system. *Computer*, 28(9):23–32, 1995.

[46] Philip WL Fong. Preventing sybil attacks by privilege attenuation: A design principle for social network systems. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 263–278. IEEE, 2011.

[47] Philip WL Fong, Mohd Anwar, and Zhen Zhao. A privacy preservation model for facebook-style social network systems. In *Computer Security–ESORICS 2009*, pages 303–320. Springer, 2009.

[48] J. Fridrich. *Steganography in Digital Media: Principles, Algorithms, and Applications*. Cambridge University Press, 2009.

[49] Fabien Galand and Gregory Kabatiansky. Information hiding by coverings. In *Information Theory Workshop, 2003. Proceedings. 2003 IEEE*, pages 151–154. IEEE, 2003.

[50] Ryan Gallagher. The Threat of Silence, 2013. URL http://www.slate.com/articles/technology/future_tense/2013/02/silent_circle_s_latest_app_democratizes_encryption_governments_won_t_be.html.

[51] 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. ACM, 2005.

[52] Daniel B. Giffin, Amit Levy, Deian Stefan, David Terei, David Mazières, John Mitchell, and Alejandro Russo. Hails: Protecting data privacy in untrusted web applications. In *10th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 47–60. USENIX, 2012.

[53] Jonathan Good. How many photos have ever been taken? URL http://blog.1000memories.com/94-number-of-photos-ever-taken-digital-and-analog-in-shoebox.

[54] Google. Closure Tools, . URL https://developers.google.com/closure/.

[55] Google. webp: A new image format for the Web, . URL https://developers.google.com/speed/webp/.

[56] Google. Software updates: Courgette, 2009. URL http://dev.chromium.org/developers/design-documents/software-updates-courgette.

[57] Saikat Guha, Kevin Tang, and Paul Francis. Noyb: Privacy in online social networks. In *Proceedings of the first workshop on Online social networks*, pages 49–54. ACM, 2008.

[58] Saikat Guha, Mudit Jain, and Venkata Padmanabhan. Koi: a location-privacy platform for smartphone apps. In *Proceedings of the 9th Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.

[59] Benjamin Henne, Christian Szongott, and Matthew Smith. Snapme if you can: privacy threats of other peoples' geo-tagged media and what we can do about it. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, WiSec '13, pages 95–106, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1998-0. doi: 10.1145/2462096.2462113. URL http://doi.acm.org/10.1145/2462096.2462113.

[60] Barron C Housel, George Samaras, and David B Lindquist. Webexpress: a client/intercept based system for optimizing web browsing in a wireless environment. *Mobile Networks and Applications*, 3(4):419–431, 1998.

[61] Daniel C Howe and Helen Nissenbaum. Trackmenot: Resisting surveillance in web search. *Lessons from the Identity Trail: Anonymity, Privacy, and Identity in a Networked Society*, pages 417–436, 2009.

[62] Ren-Junn Hwang, Timothy Shih, Chuan-Ho Kao, and Tsung-Ming Chang. Lossy compression tolerant steganography. In Won Kim, Tok-Wang Ling, Yoon-Joon Lee, and Seung-Soo Park, editors, *The Human Society and the Internet Internet-Related Socio-Economic Issues*, volume 2105 of *Lecture Notes in Computer Science*, pages 427–435. Springer Berlin / Heidelberg, 2001. ISBN 978-3-540-42313-3. URL http://dx.doi.org/10.1007/3-540-47749-7_34.

[63] Karthick Jayaraman, Vijay Ganesh, Mahesh Tripunitara, Martin Rinard, and Steve Chapin. Automatic error finding in access-control policies. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 163–174, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0948-6. doi: 10.1145/2046707.2046727. URL http://doi.acm.org/10.1145/2046707.2046727.

[64] Maritza Johnson, Serge Egelman, and Steven M Bellovin. Facebook and privacy: it's complicated. In *SOUPS '12: Proceedings of the Eighth Symposium on Usable Privacy and Security*, 2012.

[65] Sanjay Kairam, Mike Brzozowski, David Huffaker, and Ed Chi. Talking in circles: selective sharing in google+. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1065–1074, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1015-4. doi: 10.1145/2207676.2208552. URL http://doi.acm.org/10.1145/2207676.2208552.

[66] Imrul Kayes and Adriana Iamnitchi. Aegis: A semantic implementation of privacy as contextual integrity in social ecosystems. In *11th International Conference on Privacy, Security and Trust (PST)*, July 2013.

[67] Dan Kegel. The C10K problem. http://www.kegel.com/c10k.html.

[68] Robert J Kosinski. A literature review on reaction time. *Clemson University*, 10, 2008.

[69] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M Frans Kaashoek, Eddie Kohler, and Robert Morris. Information flow control for standard os abstractions. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 321–334. ACM, 2007.

[70] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, et al. Oceanstore: An architecture for global-scale persistent storage. *ACM SIGARCH Computer Architecture News*, 28(5):190–201, 2000. ISSN 0163-5964.

[71] LaCie. Wuala. http://www.wuala.com/.

[72] John Levine. *Flex & bison*. O'Reilly Media, 2009.

[73] J. Li, M. Krohn, D. Mazières, and D. Shasha. Secure untrusted data repository (SUNDR). In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation-Volume 6*, pages 9–9. USENIX Association, 2004.

[74] Jinyuan Li and David Mazieres. Beyond one-third faulty replicas in byzantine fault tolerant systems. In *Proc. NSDI*, 2007.

[75] W. Li and N. Yu. A robust chaos-based image encryption scheme. In *Multimedia and Expo, 2009. ICME 2009. IEEE International Conference on*, pages 1034–1037. IEEE, 2009.

[76] Kuan-Yu Lin and Hsi-Peng Lu. Why people use social networking sites: An empirical study integrating network externalities and motivation theory. *Computers in Human Behavior*, 27(3):1152–1161, 2011.

[77] Jørn Lind-Nielsen. Buddy bdd library, 2002.

[78] Heather Richter Lipford, Gordon Hull, Celine Latulipe, Andrew Besmer, and Jason Watson. Visible flows: Contextual integrity and the design of privacy mechanisms on social network sites. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 4, pages 985–989. IEEE, 2009.

[79] Yabing Liu, Krishna P Gummadi, Balachander Krishnamurthy, and Alan Mislove. Analyzing Facebook privacy settings: User expectations vs. reality. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 61–70. ACM, 2011.

[80] Dylan Love. 17 People Who Were Fired For Using Facebook. URL http://www.businessinsider.com/facebook-fired-2011-5.

[81] Alexis Madrigal. The Philosopher Whose Fingerprints Are All Over the FTC's New Approach to Privacy, 2012. URL http://www.theatlantic.com/technology/archive/2012/03/the-philosopher-whose-fingerprints-are-all-over-the-ftcs-new-approach-to-privacy/254365/.

[82] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish. Depot: Cloud storage with minimal trust. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, pages 1–12. USENIX Association, 2010.

[83] Evan Martin and Tao Xie. Inferring access-control policy properties via machine learning. In *Policies for Distributed Systems and Networks, 2006. Policy 2006. Seventh IEEE International Workshop on*, pages 4–pp. IEEE, 2006.

[84] D. Mazières and D. Shasha. Don't trust your file server. In *hotos*, page 0113. Published by the IEEE Computer Society, 2001.

[85] D. Mazières and D. Shasha. Building secure file systems out of Byzantine storage. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 108–117. ACM, 2002. ISBN 1581134851.

[86] Donald Metzler, Susan Dumais, and Christopher Meek. Similarity measures for short segments of text. In *Advances in Information Retrieval*, pages 16–27. Springer, 2007.

[87] Microsoft Safety Security Center. Create strong passwords, 2013. URL http://www.microsoft.com/security/online-privacy/passwords-create.aspx.

[88] Alan Mislove. Post: a secure, resilient, cooperative messaging system. 2003.

[89] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. *ACM SIGOPS Operating Systems Review*, 35(5):174–187, 2001. ISSN 0163-5980.

[90] Andrew C Myers and Barbara Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(4):410–442, 2000.

[91] Arvind Narayanan. What happened to the crypto dream?, part 1. *Security & Privacy, IEEE*, 11(2):75–76, 2013.

[92] Arvind Narayanan. What happened to the crypto dream?, part 2. *IEEE Security & Privacy*, 11(3):0068–71, 2013.

[93] Derek Newton. Dropbox authentication: insecure by design, April 2011. URL http://dereknewton.com/2011/04/dropbox-authentication-static-host-ids/.

[94] H Nissenbaum. *Privacy in context: Technology, policy, and the integrity of social life*. Stanford Law Books, 2009.

[95] H. Nissenbaum. A contextual approach to privacy online. *Daedalus*, 140(4): 32–48, 2011.

[96] Office for Civil Rights. Summary of the HIPAA privacy rule. United States Department of Health & Human Services, 2003.

[97] Ronald Oussoren, Bill Bumgarner, Steve Majewski, Lele Gaifax, and et al. PyObjC – the Python $\leftrightarrow$ Object-C bridge, 2012. URL http://pythonhosted.org/pyobjc/.

[98] Jun Pang and Yang Zhang. A new access control scheme for facebook-style social networks. *arXiv preprint arXiv:1304.2504*, 2013.

[99] Ben Parr. Facebook by the Numbers. URL http://mashable.com/2011/10/21/facebook-infographic/.

[100] Colin Percival. Naive differences of executable code. *Draft Paper, http://www.daemonology. net/bsdiff*, 2003.

[101] Ulrike Pfeil, Raj Arjan, and Panayiotis Zaphiris. Age differences in online social networking–a study of user profiles and the social capital divide among teenagers and older users in myspace. *Computers in Human Behavior*, 25(3): 643–654, 2009.

[102] Linda Fogg Phillips, Derek Baird, and BJ Fogg. Facebook for educators. *Retrieved June*, 5:2011, 2011.

[103] A. Poller, M. Steinebach, and H. Liu. Robust image obfuscation for privacy protection in web 2.0 applications. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 8303, page 1, 2012.

[104] Juliette Powell. *33 million people in the room: how to create, influence, and run a successful business with social networking*. Ft Press, 2008.

[105] Princeton University Office of Information Technology IT Security. Tips for creating strong, easy-to-remember passwords, 2013. URL http://www.princeton.edu/itsecurity/basics/passwords/.

[106] Moo-Ryong Ra, Ramesh Govindan, and Antonio Ortega. P3: Toward Privacy-Preserving Photo Sharing. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2013.

[107] M.O. Rabin. *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[108] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Pond: the oceanstore prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 1–14, 2003.

[109] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[110] Simone Santini and Ramesh Jain. Similarity measures. *Pattern analysis and machine intelligence, IEEE transactions on*, 21(9):871–883, 1999.

[111] McAfee Security. McAfee Social Protection. URL https://apps.facebook.com/socialprotection/.

[112] Somini Sengupta and Kevin J. OBrien. Facebook Can ID Faces, but Using Them Grows Tricky. *The New York Times*, 2012.

[113] Stephen Shankland. Facebook tries Google's WebP image format. http://news.cnet.com/8301-1023_3-57580664-93/facebook-tries-googles-webp-image-format-users-squawk/.

[114] S. Sheng, L. Broderick, CA Koranda, and JJ Hyland. Why Johnny still can't encrypt: evaluating the usability of email encryption software. In *Symposium On Usable Privacy and Security*, 2006.

[115] Neil T. Spring and David Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '00, pages 87–95, New York, NY, USA, 2000. ACM. ISBN 1-58113-223-9. doi: 10.1145/347059.347408. URL http://doi.acm.org/10.1145/347059.347408.

[116] Luke Stark and Matt Tierney. Lockbox: Applying the value of privacy to cloud storage. In *1st International Workshop on Values in Design–Building Bridges between RE, HCI and Ethics*, page 63, 2011.

[117] John Stevens. The Facebook divorces: Social network site is cited in 'a THIRD of splits'. URL http://www.dailymail.co.uk/femail/article-2080398/Facebook-cited-THIRD-divorces.html.

[118] Z. Stone, T. Zickler, and T. Darrell. Toward large-scale face recognition using social network context. *Proceedings of the IEEE*, 98(8):1408–1415, 2010.

[119] J. Strauss, J.M. Paluska, C. Lesniewski-Laas, B. Ford, R. Morris, and F. Kaashoek. Eyo: device-transparent personal storage. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, pages 35–35. USENIX Association, 2011.

[120] Don Tapscott. *Grown Up Digital: How the Net Generation is Changing Your World HC*. Mcgraw-Hill, 1 edition, 2008. ISBN 0071508635, 9780071508636.

[121] D.B. Terry, M.M. Theimer, K. Petersen, A.J. Demers, M.J. Spreitzer, and C.H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated

storage system. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 172–182. ACM, 1995. ISBN 0897917154.

[122] The Chromium Authors. libjpeg — The Chromium Projects. URL http://src.chromium.org/viewvc/chrome/trunk/src/third_party/libjpeg/.

[123] Niraj Tolia, Michael Kaminsky, David G Andersen, and Swapnil Patil. An architecture for internet data transfer. In *Proc. of NSDI*, volume 6, pages 253–266, 2006.

[124] A. Tridgell and P. Mackerras. The rsync algorithm. 2004.

[125] International Telecommunication Union. Digital Compression and Coding of Continuous-tone Still images. CCITT Rec. T.81, 1992. URL http://www.w3.org/Graphics/JPEG/itu-t81.pdf.

[126] Steve Vandebogart, Petros Efstathopoulos, Eddie Kohler, Maxwell Krohn, Cliff Frey, David Ziegler, Frans Kaashoek, Robert Morris, and David Mazières. Labels and event processes in the asbestos operating system. *ACM Transactions on Computer Systems (TOCS)*, 25(4):11, 2007.

[127] Zhou Wang, Alan C Bovik, Hamid Rahim Sheikh, and Eero P Simoncelli. Image quality assessment: From error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.

[128] Andreas Westfeld and A Pfitzmann. High capacity despite better steganalysis (f5–a steganographic algorithm). In *Information Hiding, 4th International Workshop*, volume 2137, pages 289–302. Pittsburgh, PA, 2001.

[129] David Wheeler. SLOCCount. URL http://www.dwheeler.com/sloccount/.

[130] A. Whitten and J.D. Tygar. Why Johnny cant encrypt: A usability evaluation of PGP 5.0. In *Proceedings of the 8th USENIX Security Symposium*, volume 99. McGraw-Hill, 1999.

[131] Guang Xiang, Bin Fan, Ling Wang, Jason Hong, and Carolyn Rose. Detecting offensive tweets via topical feature discovery over a large scale twitter corpus. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1980–1984. ACM, 2012.

[132] N Zeldovich, S Boyd-Wickizer, and D Mazieres. Securing Distributed Systems with Information Flow Control . *NSDI*, 2008.

[133] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. Making information flow explicit in histar. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 263–278. USENIX Association, 2006.

[134] Chi Zhang, Jinyuan Sun, Xiaoyan Zhu, and Yuguang Fang. Privacy and security for online social networks: challenges and opportunities. *Network, IEEE*, 24(4):13–18, 2010.

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