# Security Research for the Public Good: A Principled Approach

by

Michael A. Specter

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2021



Author . . . . . .
Department of Electrical Engineering and Computer Science
August 27, 2021

Certified by . . . . . .
Gerald Jay Sussman
Panasonic Professor of Electrical Engineering
Thesis Supervisor

Certified by . . . . . .
Daniel J. Weitzner
3Com Founders Principal Research Scientist
Computer Science and Artificial Intelligence Laboratory
Thesis Supervisor

Accepted by . . . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee for Graduate Students

# Security Research for the Public Good: A Principled Approach

by

Michael A. Specter

Submitted to the Department of Electrical Engineering and Computer Science
on August 27, 2021, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

## Abstract

Recent history is littered with examples of software vendors betraying user trust, exposing the public to exploitable code, data leaks, and invasive privacy practices. Undirected security research may be insufficient for preventing such foreseeable and preventable failures, as these problems are often the result of misaligned vendor incentives rather than the technical specifics of the systems themselves.

This dissertation illustrates the utility of security research that is motivated explicitly by the goal of realigning incentives of market actors toward providing better security. We find that a research approach guided by a deep understanding of the economic, regulatory, and technical attributes of the actors involved is crucial for solving important societally-relevant problems in computer security. We present three case studies in applying this vision:

Our first case study considers vulnerability discovery as applied to Internet voting. We perform a security analysis of the dominant Internet voting systems used in U.S. federal elections, including those used in the 2020 U.S. presidential race. We find that, despite decades of research in cryptography and voting, all deployed systems are of simplistic design and suffer basic security and privacy problems, supporting the conclusion that the market is in failure.

Our second case study involves designing cryptography to disincentivize (rather than prevent) bad behavior through the example of deniability in messaging. We find that the evolution of the email ecosystem has inadvertently resulted in most messages being nonrepudiable, incentivizing email theft and public exposure of private data. We present cryptographic constructions that solve this problem while fitting in with email's already complicated ecosystem.

Our final case study involves government requests to mandate law enforcement access to encrypted data, colloquially known as 'backdooring' encryption. We perform a security analysis of technical proposals to provide such government exceptional access, and find that they would cause untenable security and privacy risks.

Finally, we conclude with a discussion of security research as a public good, and provide direction for future work.

Thesis Supervisor: Gerald Jay Sussman
Title: Panasonic Professor of Electrical Engineering

Thesis Supervisor: Daniel J. Weitzner
Title: 3Com Founders Principal Research Scientist
Computer Science and Artificial Intelligence Laboratory

# Acknowledgments

I thank my mother, Lainee Specter, from whom I learned a love of the arts, and my father, Barry Specter, for who taught me the joy of engineering, computer science, and history. This interdisciplinary attitude is what originally drove me to study everything I could throughout my academic career, ultimately resulting in the work you are reading now. In short, mom and dad, I blame you!

I thank my high school and undergraduate advisors, David Kosbie and David Grier. It is from these early mentors that I learned that teaching itself could be an art form, and for that I am eternally grateful.

I thank my PhD advisors Gerry Sussman and Danny Weitzner, from whom I have learned a tremendous amount, both technically and personally. Gerry Sussman taught me the value of a proper programming language, but also the need for a well engineered bike, fantastic tea, and to doggedly focus on helping students do the best work that *they* wish to do. From Danny Weitzner I learned what it takes to work with intention, to do technically-focused policy work, and to be more effective in moving a discussion forward. Being mentored by these two has most certainly been the greatest honor of my life.

I thank the rest of my thesis committee, Ron Rivest, Matthew Green, and Joan Feigenbaum. This dissertation was tremendously improved from their input, and the mentorship and guidance they provided was invaluable.

I am incredibly thankful to have the support of my many collaborators — my co-conspirators from across industry, civil society and academia. During my time at MIT I have been incredibly lucky to work with many of my heroes, some of whom the average reader is likely to recognize already, and I have no doubt that the world will come to know the rest soon: Sunoo Park, Jenny Blessing, J. Alex Halderman, Jonathan Rudenberg, Aryana Ensafi Halderman, Nakul Bajaj, James Koppel, Andy Sellars, Daniel Austin, Zara Perumal, Joel Galenson, Joshua Hodosh, Ivan Lozano, Neha Narula, John Callas, Harold Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Susan Landau, Peter G.

Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, and Julie Sussman.

I thank the team at the BU/MIT Technology Law Clinic led by Andy Sellars, Tiffany C. Li, and students John Dugger, Quinn Heath, and Eric Pfauth. Without this fantastic group's advice, patience, and effort, much of the work in this dissertation would never have been released. It is not an exaggeration to say that this work has managed to frustrate the efforts of the FBI, DOJ, GCHQ, multiple state governments, two multimillion dollar startups and their investors — keeping us out of trouble was no easy task.

I want to thank my wonderful friends Stuart Babcock, Bradley Sturt, John Hess, Doug Pietzrak, Danielle Gordon for keeping me sane during my PhD, and Dr. Linda Goodman and Dr. Daniel Bryan Goodman, whose infectious love of science and discovery convinced me that I would enjoy a PhD in the first place. I owe more to my close friends than I'll ever be able to repay.

I thank my fellow grad students, whose companionship and commiseration during my PhD reminded me that I was not alone, and whose work and influence continues to inspire me: Alin Tomescu, Leilani Gilpin, Jonathan Frankle, Ben Z. Yuan, Julius Adebayo, Cecilia Testart, Kyle Hogan, Alin Tomescu, and Derek Leung.

Thank you to the fantastic administrators Mel Robinson and Marisol Diaz, to whom the members of our group at MIT are all, ultimately, accountable.

Thank you to my partner, Cristina Logg, for sharing this experience with me, putting up with my eccentricities, and reminding me to find the joy in life, even through the most difficult of times.

Thank you to the fantastic doctors and staff at Dana Farber, whose research and care saved my partner's life, and, by extension, mine as well.

And, finally, I thank you, dear reader, for taking the time to read this work, and to all who wish to fight for security, privacy, and democracy.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

We entrust digital systems with incredible control over our daily lives, surrendering our finances, sensitive health data, constant location records, and our most personal communication. It is unlikely that this trend will reverse — luminaries studying human computer interaction have long predicted that digital systems will become far more ubiquitous, "...weaving themselves into the fabric of everyday life until they are indistinguishable from it" [313].

Unfortunately, the implicit trust we place in these systems is often broken, as their vendors repeatedly create insecurity as the result of foreseeable and preventable failures. Service providers suffer preventable data breaches by carelessly skipping patches that mitigate long-known vulnerabilities [50], mobile phone vendors sell devices with malware pre-installed to unsuspecting users [136], and the firms we trust with our most closely-held interpersonal communication willfully violate user privacy for economic gain [88]. A recent large-scale study of consumer IoT system firmwares confirmed that many deployed systems contained known-exploitable code, baked-in private keys, and a lack of basic compiler mitigations [91], while another 15-year longitudinal study found that manufacturer updates to IoT systems were more likely to *remove* security protections than add them [95]. And, when devices finally reach do reach the end-user, misinterpretation of the systems' guarantees or laziness in user implementation often result in weak or outright broken security [18].

So, why do such *avoidable* failures keep occurring? The answer is likely not wholly

technical, as the presence of these issues themselves indicate causes extrinsic to the technical specifics of the system in question — if security and privacy were valued, use of mitigations would likely be close to universal rather than (apparently) regressing! In fact, it is the growing consensus of the academic community that many vulnerabilities are caused by software vendors and users rationally choosing not to invest resources into securing their systems [19, 20, 163].

The growing societal importance of these systems further complicates the discourse surrounding security and privacy. Increasing use of encryption, for example, has afforded users greater security and privacy, but has led to law enforcement panic over loss of access to suspects' data [3]. Users have been provided a greater ability to communicate, but may not understand that their messages in many cases are not entirely private, or that when shared they often carry an irrefutable proof of their authenticity [278]. Even functions core to the inner working of democracy — the machinations of elections and process of voting itself — have been increasingly digitized to allow for greater access, but the associated uncertainty over these systems security has led to distrust, allegations of fraud, and very real attacks [277, 279]. To better protect users, we must begin to navigate both the social complexity of these problems and incentivize better behavior by those we trust to manage our systems.

## 1.1 The Need for an Interdisciplinary Approach

Academic security research does not often confront these more complex, societally-relevant topics, as much of applied security research is motivated by a set of somewhat general and self-explanatory goals. A prototype system may be developed, for instance, to provide new and useful protections given some specific problem and setting with obvious utility to a hypothetical user. Other work may focus on the discovery of novel classes of vulnerabilities, or demonstration of widespread failures of protections, providing insight into the state of systems security and motivation for further study. These approaches can be very useful, but are also inherently limited, as neither are likely to be able (or, indeed, attempt) to grapple with wider societal challenges, nor

are they capable of explaining foreseeable failures.

The inadequacy of this context-oblivious approach has not gone unnoticed in the academic community. Recent essays by Kamara [181] and Rogaway [262] each admonish the cryptography research community for systematically failing to address societal concerns. In particular, Rogaway argues that academic security research has fallen prey to overly aesthetic "puzzle solving," while Kamara laments how industry-oriented applied cryptographic research has become. Both Rogaway and Kamara argue that the community should begin to favor research that focuses on helping to prevent social ills, like unchecked mass surveillance and exploitation of marginalized communities.

However, their analyses largely stop there — while these works do an excellent job of motivating the need to provide societally-oriented technical work, there is very little insight as to *how* such research should proceed beyond a select few cases.[1] It is imperative, then, that we expand on Kamara and Rogaway's work, and consider how to engage in a more rigorous and generalizable way.

## 1.2 This Dissertation

This dissertation illustrates the utility of security research motivated explicitly by the goal of *realigning incentives of market actors toward providing better security*. Or, put in other words, that a research approach guided by a deep understanding of the economic, regulatory, and technical attributes of the actors involved is useful for solving important societally-relevant problems in computer security. This concept is explored through a series of novel technical works that, without this holistic understanding of the larger context, a researcher would be unlikely to follow. This has included security analyses that expose failing practices of vendors, the creation of cryptographic protocols geared towards disincentivizing (rather than preventing) bad actors, and research explicitly geared toward providing timely and relevant policy advice to regulators.

[1] See [124] for further critique on Rogaway.

We begin in Chapter 2 with a short primer on prior work on economics and law in information security, and introduce some of the relevant concepts in the literature used in other chapters in this work. This also serves to provide further context, background, and motivation for the rest of the work. Specifically, we provide an in-depth analysis of the interplay between information asymmetry, security, and the (somewhat novel) concept of systems security as a *credence quality* of digital products — a designation that helps explain why such preventable failures are so common, and provides further justification for leveraging incentive alignment as a guiding principle of applied systems security research.[2]

In Chapters 3-5, we demonstrate the effectiveness of this framing through three case studies. Each chapter includes an explanation of the policy environment, followed by the technical work influenced by this understanding. A brief introduction to each chapter is presented below:

**1) Internet Voting.** Our first case study involves a problem core to the democracy: the market for electronic voting systems.[3] We explore the regulatory regime and economic incentives surrounding electronic voting systems in the U.S., and find evidence that the market suffers from a myriad of issues including weak standards and regulation, limited transparency, principal agent problems, and significant information asymmetries between the purchasers of these systems and vendors, which are further complicated by the vendor to hiding or misrepresenting these systems' security properties.

To confirm our suspicion that these failings would incentivize poor security, we performed a comprehensive security analysis of OmniBallot and Voatz, the two dominant Internet voting systems deployed in U.S. federal elections. We discovered that both systems suffered from a number of egregious security and privacy violations, including attacks that would allow a variety of adversaries to reveal, alter, or stop

[2]This section draws on a larger, in-progress, joint work with Andrew Sellars.

[3]This dissertation includes examples of flaws in voting systems deployed in real, high-stakes federal elections in the U.S. and abroad. The author would like to emphasize that nothing in this dissertation should be taken to indicate that any previous election was called incorrectly, and stands by the letter he signed along with 58 other security researchers affirming that there has been no convincing evidence of result-altering malfeasance in the 2020 U.S. federal election [7].

a voter's private ballot or otherwise control the outcome of an election. These flaws were not errors of implementation, but of *design* — both systems were incredibly simplistic, with neither leveraging the decades of research available on cryptographic voting mechanisms.

The results in this chapter first appeared in "The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in US Federal Elections," and "A Security Analysis of the Democracy Live Online Voting System," which were presented at *Usenix Security '20* [277] and *Usenix Security '21* [279], respectively. The work on the Voatz system was joint effort with James Koppel and Danny Weitzner, while the investigation of Democracy Live was with J. Alex Halderman. Finally, the chapter's background on the systems requirements of voting (Section 3.1.2) was adapted partially from "Going from Bad to Worse: From Internet Voting to Blockchain Voting" [236], joint work with Sunoo Park, Neha Narula, and Ron Rivest.

**2) Deniable Email.** We continue in Chapter 4 with technical efforts to reintroduce deniability to the email ecosystem. Modern end-to-end encrypted messaging systems like Signal or iMessage use a number of methods to ensure that their messages are *deniable*, meaning that any future thief of these messages cannot prove to another party that the messages are real. It might be surprising to learn that email, a largely unencrypted medium, is not deniable, which has in-turn had real-world impact — most notably enabling WikiLeaks to publicly verify and publish a trove of stolen email provided to them by Russian intelligence during the 2016 U.S. Presidential Election. However, nonrepudiability of email has been an open problem in the literature since Off The Record's deniable key exchange protocol was first introduced fifteen years ago [62].

Our key insight is that the goal of deniable messaging is largely unlike most security mechanisms in that its utility lies solely in its ability to *disincentivize* rather than *prevent* malicious activity. Conversely, a lack of deniability incentivizes malicious behavior: With cryptographic proof, an attacker can more easily sell stolen messages to third parties, blackmail the user, or publicly release sensitive communication with

a high level of built-in credibility. This understanding provided more degrees of freedom in our implementation than previous attempts, and motivated us to tackle this particular problem, while the fact that this problem persists despite this property being the result of human readable standards supports our argument that information asymmetry in systems security is incredibly difficult to overcome.

This chapter is largely adapted from "KeyForge: Mitigating Email Breaches with Forward-Forgeable Signatures," with Sunoo Park and Matthew D. Green, which first appeared in *Usenix Security '21.*

**3) Encryption, Surveillance, and Key Escrow.** Our final case study (Chapter 5) presents a Systematization of Knowledge-style analysis of the impact of key escrow mechanisms on the security of the software ecosystem, as well as the rule of law, economy, and moral leadership. We find that all known key escrow mechanisms would force a reversal of current technical best-practices, and require solving likely insuperable technical, jurisdictional, and regulatory concerns.

While previous case studies focus on directing security research to identify current flaws or create new systems, this final chapter explores how one may leverage an understanding of technical security measures to *prevent* incoming regulation that would invariably misalign vendor incentives and introduce future vulnerabilities. This work demonstrates the importance of a melding techniques, in both understanding the societal and technological context.

Chapter 5 is adapted from "Keys Under Doormats: Mandating Insecurity by Requiring Government Access to all Data and Communications," updated to include the impact and results following the publication of the work, and supplemented with historical information to contextualize the setting. This was a joint effort with Harold Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, and Daniel J. Weitzner.

The findings in this dissertation have already had significant real-world impact. As a direct result of our voting work, many jurisdictions altered or cancelled use of

these systems for the 2020 primary and general elections, and the research was at the center of the analysis in an Electronic Frontier Foundation-led Amicus Brief to the U.S. Supreme Court [92]. While standardization is still in-process, Jon Callas, an original author of the IETF specification we critiqued as causing trouble for email, has since publicly supported [73, 74] implementing our deniability protocol, and our work has ignited debate on the issue, and was covered in the popular press as "The right to be forgotten for email" [132]. Finally, as a result of our work on encryption and surveillance, Congress's impending legislation was halted, a bipartisan House-Senate working group adopted our analysis [146], and a number of technical and policy proposals have followed.

# Chapter 2

# Background: Information Security, Economics, and Law

This chapter presents a short introduction to concepts in law, economics, and information security that will be used in many other sections of this dissertation to motivate security and privacy research. We introduce the concept of security as a *credence quality* of digital systems, a definition in economic theory helps explain why preventable failures occur, and provides justification for considering incentive alignment as a guiding principle of applied systems security research. In other words, the goal of this section is to argue that security research should be seen as a tool for realigning incentives and ameliorating information asymmetries in market economics as a part of a larger system of consumer protection.

We begin in Section 2.1 with an introduction that motivates the need for a new legal and economic framework for security research. Continuing in Section 2.2, we argue that it is necessary to update the existing literature to include an analysis of information asymmetry, and provide a short explanation of the relevant works in economics and law. Finally, we discuss how this updated understanding allows us to take lessons from other sectors and apply them to regulation of software for improved security (Section 2.3), and conclude in Section 2.4 with a discussion of how this framework helped guide the security research presented in other sections of this dissertation.

## 2.1 Introduction

A core complication in regulating software as a market is the deeply *asymmetric* relationship between the user and software vendors — individual users cannot be expected to understand how these systems work, let alone defend themselves from a developer's (potentially) malicious behavior. Take, as a simplifying example, a user's commodity smartphone. In practice, any such device represents an immensely complicated system developed and maintained by a myriad of software and hardware vendors, standards organizations, and various other third-parties. The result is complication to the point of inscrutability; it is unlikely that any one *institution* can have a full understanding of the components of the end system, let alone a user. Even assuming one were to get full access to schematics, code, and design documents of their device, they would *still* need an expert level of time and expertise to perform an audit capable of estimating if the developers had followed best practices.[1]

The reader might find it surprising that the academic legal literature that studies security, and particularly the process of adversarially understanding a product through reverse engineering, does so mainly through the lens of interoperability, innovation, intellectual property, and competition policy. For example, the leading analysis of the legal treatment of reverse engineering comes from Pamela Samuelson and Suzanne Scotchmer in 2002 [266], which addresses a wave of protectionist intellectual property legislation and international trade agreements that have placed the practice of reverse engineering under threat.

As Samuelson and Scotchmer put it:

> "[T]he legal rule favoring reverse engineering in the traditional manufacturing economy has been economically sound because reverse engineering is generally costly, time-consuming, or both. Either costliness or delay can protect the first comer enough to recoup his initial research and development (R&D) expenditures. If reverse engineering (and importantly, the

[1] Indeed, as we will see in Chapter 4, completely open standards are still so opaque to everyday users as to present steep barriers to common sense understanding of the system's security and privacy guarantees.

> consequent reimplementation) of manufactured goods becomes too cheap or easy ... it may be economically sound to restrict this activity to some degree." [266]

As indicated by their caveat about "reimplementation," key in their analysis is that the end goal of the reverse engineer is to implement a competing product, and that any restraints on reverse engineering should be very narrowly tailored to "parasitic activities" such as "market-destructive reimplementations of innovations." Samuelson and Scotchmer do later indicate some broader public benefits, but their focus, as with others, remains on impact reverse engineering will have on the incentive to create new technologies and works.

Other reverse engineering scholarship similarly focuses its analysis on the development of new and interoperable software [35], as do the courts.[2] Even those who argue for removing restraints on reverse engineering tend to do so through the lens of innovation theory, or some general sense that one should have the "freedom to tinker" with their own purchases, and not with any broader recognition of the heightened public role that software plays in modern life and how reverse engineering can help hold system vendors accountable.

## 2.2 Expanding the Legal & Economic View of Security Research

Such competition-based framing fails to account for the public need of expert analysis to protect consumers from malicious, fraudulent, and privacy invasive software. Though analysts often gain some indirect or limited remuneration (e.g. through academic and journalistic publishing, peer recognition, or a bug bounty payout), or direct remuneration (e.g. via a contract consulting engagement), many did so without the intent of creating new or competing products at all.

[2] See, e.g., Bonito Boats, Inc. v. Thunder Craft Boats, Inc., 489 U.S. 141, 160–61 (1988) [61] (noting that reverse engineering induces new product innovation and "may act as a spur to the inventor, creating an incentive to develop inventions that meet the rigorous requirements of patentability").

However, similar to Samuelson and Scotchmer, we find that one need not depart from an economic analysis to explain an expanded role for reverse engineering. In fact, we must do little more than incorporate what has already been said on the subject by those in the computer security, and law and economics communities (albeit separately).

The computer security literature has repeatedly discussed software security and privacy as an economic problem largely stemming from misaligned incentives, vendor lock-in, and the asymmetric relationship between the user and software vendors [19, 20, 220, 19]. At the intersection of Human-Computer Interaction (HCI) and security, researchers have found that users often rationally choose not to follow security advice from experts due to the time and effort doing so would require [163], often do not understand the security and privacy properties of the systems they use [182], and that many available techniques are so difficult to use that they are practically worthless [317].

In a similar vein, the economics literature has much to say about markets for products that suffer from *asymmetric information* – in which sellers have important information not known by purchasers. As will be explained in later sections, these information asymmetries generally lead to bad outcomes for the purchaser, and, at the worst, can result in an entire market for "good" products failing to exist at all. The goal of many proposed regulatory interventions in classic welfare economics is to improve efficiency by ameliorating or removing this asymmetry, or by forcing the market to consider other values that are in the public interest.

To aid in their analyses of the markets for different products, economists often find it helpful to split goods into three categories according to the difficulty of mitigating the information asymmetry between purchasers and sellers: These are search, experience, and credence goods. These are not hard-cutoff definitions, but a spectrum describing the amount of asymmetry that exists in the market. Indeed, some parts of a product might be more asymmetric than others, such that good might be partially an experience good and partially a credence good. In this case, we would describe this as a product with a *quality* that is of a particular type, e.g., a "credence quality." We briefly explore each category below and discuss their relevancy (or lack thereof) to software security.

### 2.2.1 Search Goods

Search goods (or qualities) are products (or aspects of products) whose value is obvious from their inspection, leading buyers and sellers to have little-to-no asymmetry of information. For example, a purchaser will likely know almost everything about a nail before it is finally purchased — the size, weight, melting point, and material makeup of the physical object is easily understandable, measurable, and standardized. Buying clothes follows a similar formula, as a buyer can usually try the product on, test it, and understand if the garment is right for them before actually finalizing the deal.

The upshot of search goods is that they are often the simplest to regulate. The purchaser has little difficulty knowing what product they need, and the seller would have a difficult time scamming their customers. As a result, the greatest challenge is determining market price, which requires (potentially) strenuous legwork and analysis by the buyer, but one does not need to buy the product to understand its value.

Unfortunately, there are qualities of software, especially of larger and more complex systems, that do not fit the definition of a search good, as understanding how a software-based product functions before it is purchased is difficult, if not impossible. It is often not possible for a user to fully understand the limitations, usability, and functionality of a software product from a short, in-person display or quick use-case; failure modes are often latent and only discovered after significant use.

Worse, modern software is more akin to a sustained service than a one-off purchase of a product [166]. In a way very unlike, say, a garment or a nail, software ages as external pressures such as the user's other hardware, software, and operating system evolve; over time, software *rots*[3] requiring constant maintenance by the vendor via patches to sustain functionality, interoperability, and security. As a result, a need to repair and replace components of the system will almost always be necessary. In short, a program that was useful, secure, and otherwise non-harmful today may not be three weeks from now, and the distinction relies on how the developer chooses to provide

[3]"Software rot" and "bit rot," are ideas in the discipline of software engineering largely attributed to folklore, though they appear to originate from a the Jargon File, a compendium of software engineering terms compiled by operating systems engineers at MIT, Stanford, and Carnegie Mellon in the mid 70's and early 80's. See, generally, [253]

maintenance – a factor that cannot be known without purchase of the product.

### 2.2.2 Experience Goods

So, since usability, long-term support, and other factors cannot be determined before the product has been purchased, it is worth considering if software may instead fall into the category of an *experience good*; a product that requires purchase and use before the item's true value can be known. Experience goods markets can be a bit tricky, as purchasers must price in the likely cost of whatever imperfections exist in the purchased product. This, in turn, may result in the market being naturally non-viable and require government intervention to avoid systemic failure.

Perhaps the most famous work on experience goods is Akelrof's Market for Lemons [13], which argues that the inability for purchasers to understand what they are buying can easily lead to "lemons markets" in which good products are systematically driven out in favor of cheaper and poorer alternatives. Akelrof uses the market for used cars as a thought experiment to explain the concept. In his model, there are "lemons," used cars that are below the market rate due to some sort of issue with the car, and "cherries," which exist at or above the market rate. As rational actors, the sellers want to maximize profit, and should not sell their car below the market rate; all or most owners with vehicles worth more than the market (cherries) will refuse to sell. Purchasers inevitably notice that the average quality has decreased and will reply by lowering the price they are willing to pay, thus driving the market rate for these cars downward, resulting in more of the market's cars to be above the average market price line. This process repeats until only the bottom-of-the-barrel "lemons" are left.

This concept, and its parallels to what has been happening in the software security world, has not gone unnoticed by those in the academic computer security community. Perhaps the most well-cited paper on the economics of information security is Anderson and Moore's work, which explicitly references Akelrof:

> "In some cases, **security is even worse than a lemons market**: even

> the vendor does not know how secure its software is. So buyers have no reason to pay more for protection, and vendors are disinclined to invest in it." [20] (emphasis added)

What Anderson and Moore are grappling with is the difficulty of categorizing software as a product. By calling software "worse than a lemons market," they are arguing that, though it does appear to have the race-to-the-bottom problem of a lemons market, software security still does not fit neatly into the definition of an experience good, as it is a much higher bar to fix the information asymmetry involved. Indeed, if vendors *themselves* will not expend the extra time and capital to understand these properties themselves, how could they explain it to their *users*?

There are a few reasons that software cannot be easily defined as an experience good. Many properties of software, including its security and privacy features, are largely imperceptible to non-expert users and require painstaking review by experts to assess. Take, for example, a videoconferencing suite's failure to implement end-to-end encryption: The average user cannot perform the analysis necessary to understand if the system is end-to-end encrypted,[4] and, in the case that the user understands the concept itself, few will take the time or effort to instrument the software to figure out its limitations.

### 2.2.3 Credence Goods

When purchasers cannot adequately evaluate a quality of a product (or the value of the product itself) despite having purchased an had first-hand experience with its use, it is classified as a *credence good*. This is not to say that the value of credence goods or qualities are never known, just that understanding the value of the product instead requires some expert knowledge or analysis [97].

In many contexts, software behaves like a credence good, as users often cannot

[4]Indeed, "end to end" encryption is itself somewhat of an imprecise term based on the expectation of the user rather than having a technical definition in and of itself. For example, to which "ends" is the communication encrypted? Just between the user and the server, or between the user and all other users? The original definition of End to End comes from a seminal work in networking that refers mainly to client-server interaction, though the cryptographic definitions often refer to users.

know much about how the system works. Even if some properties of the system are understandable to a user a priori (such as UI design, speed, some specific functionality, etc), security properties themselves are certainly almost impossible to assess by the average user and therefore represent credence qualities.

Worryingly, the need for this expensive analysis often leaves credence goods markets rife with fraud, as this intense information asymmetry can incentivize unscrupulous vendors to swindle their users with limited chance of detection. A classic example is a repair man: A customer may ask for a repair to be done, but, cutting corners, a contractor might use cheaper parts than necessary while charging the purchaser the full price. In the future, even if the same exact item requires repair again due to the previous repairman's shoddy work, the purchaser may still never know if this failure was caused naturally or by fraud. In this case, the repairman is *undertreating* their customer by providing some weaker product than is required, but an unskilled buyer may never know.

And, just as a repairman can undertreat, so too can software developers. Failing to deploy security patches, respond to vulnerability reports by well meaning external researchers, apply unobtrusive compiler-based mitigations, use type-safe programming languages, or properly encrypt data are all instances of vendors potentially undertreating the security quality of their products. This sort of undertreatment appears to be a nontrivial factor in software security in particular: A recent large-scale study found 38 previously unknown vulnerabilities in over 600 device firmware images affecting hundreds of thousands of internet-accessible devices, while leveraging only automated, non-novel, free, and openly available vulnerability analysis tools [91]!

It is also worth noting that the converse can also happen – a disreputable expert could overtreat, providing a stronger, more costly treatment than what is required, to "upsell" the buyer. For example, a shifty car mechanic might suggest that it is necessary to replace an unsuspecting customer's entire engine, when only a part needs to be fixed. Similarly, doctors often cannot undertreat patients for fear of malpractice, especially if the undertreatment would result in an obvious failure, but may instead opt to overtreat to boost their income. This seems to have been proven empirically;

studies have shown that, on average, doctors are less likely to prescribe costly medical procedures to a patient if that patient is also a doctor [120].

## 2.3 Regulating Software Security as a Credence Good

So, what does this definition of software security as a credence good tell us about how one should think about reverse engineering, security research, or software regulation in general? Unfortunately, this designation indicates that there are no simple answers — the economics literature explains that it is difficult to regulate credence good markets to maximize social welfare.

This is not to say that it is impossible to provide workable regulation in these settings. Services such as those provided by doctors may require extensive training, certification, often have to carry liability insurance. Repair services at the high end are often mediated by insurance companies that provide third party expertise on the needs of the consumer (e.g. if you need body work on your car, your insurance company may set a price that must be met by a body shop). Governments and third party payers may force these kinds of services to be subject to extensive price and quality controls. None of these systems are perfect, but they are strategies that are used to manage credence good challenges.

More formally, fraud in credence goods markets are often controlled through two main factors: liability and verifiability. Many works consider other factors, like homogeneity of purchasers, the ability to split diagnosis and purchasing costs (as in our earlier example of insurance setting prices), and other factors. For pedagogical purposes we focus on these two properties, as they have been proven to be the most generalizable.

In the economics literature, *liability* refers broadly to mechanisms that make vendors unable to provide a weaker treatment than what is necessary. In the legal sense, it represents the concept that one is obligated to behave in some way, under penalty of suit, tax, or other disincentive. Such regulation can be a strong incentive not to undertreat a consumer, but does not change overcharging or overtreatment.

*Verifiability*, conversely, is more concerned with the user's ability to observe and demonstrate to the courts the value of the treatment that they received. The idea is that one cannot charge for an expensive procedure and then provide a cheap one instead, which prevents overcharging and overtreating. From these descriptions alone we can already begin to understand why certain regulatory attempts may fail to provide adequate cover for users. For example, a naive rule for critical code might require the vendor to divulge proprietary information about the product to allow for third party review. It is often the case that systems' source, algorithms, and other sensitive information may need to be shared to provide worthwhile guarantees that the system has been, or can be, adequately examined by experts.

However, the exposure of this sensitive data alone may still be unhelpful. Though exposure of the system's source may lower cost of expert analysis, providing this guidance is still often time consuming and difficult, and there may still be no incentive for researchers to do so. Worse, without any sort of liability, there may be no incentive for software vendors to modify their code given this expert guidance.

Finally, many of the questions that would determine negligence or liability require careful analysis of the system involved, a larger understanding of the software ecosystem, and an understanding of the intended use of the system. For example, how does one accurately judge if a product's flaws are due to honest error, or a vendor's lack of investment? And how much should we care if one system fails to apply a particular set of mitigations, while another succeeds? What are the user's expectations of privacy, and the state of the art in securing this particular kind of system?

Answering these sorts of liability questions is often incredibly difficult, even with a full understanding of the system itself, as users' security needs are also *non-homogeneous*. For example, the device and software used for basic email and web-surfing may be identical to what is being relied upon by heads of state for mission-critical communiques. Even in the simplest case of one software vendor that is aware of the security and privacy requirements of the consumer, the constituent subsystems, libraries, and platforms on which the system is built are likely not involved. If a device or service that is provided with an intended user-base is instead used in security-critical

situations, should *that* vendor be expected to provide that security? How?

It is worth noting that these sorts of complications explain the failure of transparency alone to provide substantive security guarantees (as argued by Kroll et. al [189]) — security failures are usually a symptom undertreatment and cannot be helped by only making software more verifiable without allowing for any recourse.

More alarmingly, a game theoretic analysis by Dulleck and Kershbamer [111] indicates that a market lacking homogeneity, liability, and verifiability will trend toward a lemons-market-style failure. Follow-on work leveraging behavioral and experimental analysis further confirmed this finding [112], but also hearteningly found that, even without legal liability, roughly half of sellers will be honest. Presumably, to these sellers, social and moral norms may outweigh increased economic benefit. Other works consider the idea of a vendor being able to build a reputation as an honest broker, but studies have found that, in the absence of verifiability and liability, these reputation effects help increase the volume of sales, but have little impact on vendors propensity to cheat [119].

Less can help if the user lacks a motive to care about the security and privacy properties of the systems they use, and it is often the case that this assumption does not hold. In a practical sense, the consequences of a breach are often *externalities*, as the negative result is suffered not by the owner of the vulnerable or exploited device, or the vendor of the insecure code, but by another, unrelated third party. A botnet that has infected millions of IoT devices, for example, may do nothing to harm the system's owner, but instead be leveraged for debilitating distributed denial of service attacks against other services [24].

Finally, it is worth considering the applicability of software licences as a partial solution. In particular, this credence good argument provides a practical backing to the need for *free software* licences, which ensure that users "have the freedom to run, copy, distribute, study, change and improve the software" [315]. As the source of such software is inherently open for analysis, and there is a sort of limited liability in that a failure to update or fix vulnerabilities may result in third parties copying, altering the code, and creating their own version, such licences do provide some protections

to the user. A notable example involves the history of the free cryptography library OpenSSL, which, after a critical vulnerability was discovered, had multiple parties creating their own versions of the software, each of which having significantly lower complexity and a decreased number of vulnerabilities as compared to the original [57].

### 2.3.1 Labeling: Lessons from Environmental Regulation

In 2021, Apple began requiring developers who deployed software through Apple's app store to publicly list what information the app collects about the user. This "nutrition label" is then presented to users before downloading the app. Remarkably, these labels are self-reported, and are not verified by Apple in any way; they do not prevent the application developer from lying and hiding details about what data they collect. So, why would Apple do this at all, and is there any hope of these labels improving user privacy?

To better understand this tactic, we turn toward related work in economics and environmental regulation. Like security, the environmental impact of a product is a credence quality; one cannot know if a product has been made in some environmentally sustainable way, though consumers might value and factor this property into their purchasing decisions. As a result, companies often label their products with statements confirming their good standing, such as "dolphin safe tuna," "EnergyStar," and "free trade." These monikers are called "Ecolabels," and purchasers may choose to buy these products at a markup over other, unlabeled products of similar quality [80].

The economics literature has extensively studied labeling, finding various regimes effective depending on the specifics of the market. This has included paying or collaborating with third parties to affix labels and rate products, self-labeling without outside intervention, through industry standards organizations, and via mandated compliance from the government. In each of these cases, there have been incidents of successes and failures, largely dependent on the system's ability to avoid unenforceable, arbitrary claims by vendors. These inaccurate or worthless claims might look like meaningful concessions to purchasers, but effectively *increase* information asymmetry, and may induce purchase of a harmful good under false pretenses.

Luckily (at least, for the U.S.), many Ecolabels have defined meanings that the U.S. government's Federal Trade Commission (FTC) can enforce [149]. The FTC typically initiates an enforcement action after receiving a reason to believe that the law has been violated, often following an investigation. These investigations frequently begin by other legal bodies or the public raising an issue with the FTC, often through contacting the Commission via a signed letter, or the FTC's Consumer Sentinel Network. Assuming there is reason to do so, the Commission then negotiates and settles virtually all of these cases, with the company usually agreeing to follow a legally binding guideline called a *consent decree* designed to ensure ongoing compliance [165]. False labels appear to be a pernicious problem in computer security. Many software vendors use nonsense terms like "doubly anonymized" or the insignificant "military grade encryption" (See, e.g., this dissertation's Chapter 3 on voting for a worked example). And, though the FTC has guides that explicitly define what certain *environmental* words mean, there currently exists no worthwhile equivalent for security claims.

Despite not providing explicit guidance, the FTC has had significant successes in enforcing security and privacy labels. The FTC's Enforcement actions have included fighting "deceptive" trade practices based on specific claims of security that the company failed to implement [170], more general claims based on insufficient notice of invasive activities [325], as well as through a company's more general claims of "reasonable" or "industry leading" security practices that the FTC found to be lagging or unreasonable [169]. Privacy enforcement is replete with FTC actions based on broken promises, to the point that virtually every major online service operates under a consent decree from an FTC.

Unfortunately, the FTC's ability to enforce accurate labeling is still limited. While the FTC retains notable technical experts that provide additional analysis and public communications for their work, the Commission operates with a relatively small staff and budget. It has the technical power to investigate, but despite considerable scholarly calls for the FTC to take a more proactive role in investigating online platforms, it tends to take a reactive posture to launching investigations and actions [305]. Scholars

have noted the importance of capturing the interest of FTC staff in order to drive action, and some of the original economic works on credence goods express doubt that regulators will be capable of keeping up without aid [97].

### 2.3.2 Security Researcher as "Information Supplying Activist"

All of this leads us to the utility of independent security research as a tool for incentivizing good behavior by firms. Specifically, it has been found that credence good markets devoid of liability or verifiability can benefit significantly from activists exposing a company's failure to comply with the labels they have professed to employ [34]. Critically, avoiding "cheap talk" of security requires outside accountability through reverse engineering and security analysis in order to provide verification of vendor's labeling, opening them up to liability by regulators and competitive disadvantage when reputation is important. Further, it is likely that such research can raise the effectiveness of enforcing normative label values (e.g. that end-to-end encryption of a video call indicates that the communication is encrypted between users, and not the server and the user), which can hopefully lead to formally defined labels and other regulatory enforcement.

For example, the FTC has often relied heavily on public reporting and academic security research as a means of determining where to dedicate its limited resources. This can be seen most recently in its action against the videoconferencing suite Zoom [171]. As evidence in its complaint, the FTC cited a public statement about the company's encryption configuration by Zoom's Chief Product Officer, Oded Gal [134]. This statement was made a day after Micah Lee and Yael Grauer's independent examination of Zoom for *The Intercept*, apparently in direct response to what Lee and Grauer had found [197]. Dissenting Commissioner Rohit Chopra specifically cites *The Intercept*'s reporting in arguing that the FTC should have imposed a greater sanction against Zoom. In a similar dissent, Acting Chair Rebecca Kelly Slaughter cited other independent research, including research into Zoom sharing LinkedIn information without consent and that Zoom's cloud servers had a predictable URL structure which would allow others to guess where meeting recordings were saved.

Independent research has served as an input in a number of other FTC enforcement actions. The FTC's action against Snapchat for falsely suggesting that messages disappeared permanently was informed by security research that inspected the temporary file system of the Snapchat application and discovered that "disappearing" images were in fact stored therein [170]. Independent security research firm Kryptowire performed a mixture of static and dynamic analysis to reveal that BLU Products, Inc. shared information that went beyond the scope of their privacy policy [228], leading to an FTC action [167]. Researchers at Stanford informed an FTC action against Epic Marketplace by discovering that the company engaged in "history stealing" on third-party websites, in contravention of its privacy statements [209, 168]. As more and more attention is paid on consumer protection issues in technology, this form of independent accountability will no doubt increase.

This form of research will no doubt play an even more significant role as the FTC broadens its privacy enforcement from cases involving exposure of data and security into broader notions of computational fairness and policing against bias in software outcomes. Studies around disparate algorithmic treatment, be that bias within tools used for employee hiring and recruitment [303], or race, gender, and intersectional bias within software tools like facial recognition [68], are all likely to inform any action the FTC decides to take in this area. This form of examination not only provides the ground truth of how the algorithms behave, but does so in a way that's readily understandable by regulators. Ruha Benjamin has noted that for audits of this nature to be successful, they must be both independent and enforceable [48].

Remarkably, even if one is skeptical of the FTC and government intervention in general, there is still evidence that security research can help. An economics paper co-authored by the current head of Stanford's Hoover Institution argues that "information supplying activists" can help regulate credence good markets by stochastically exposing malfeasance [123]. Their model indicates that, in the presence of this kind of adversarial relationship, the market equilibrium will move firms toward strategically choosing to adopt welfare maximizing practices as a means to avoid the harm inflicted. This is further explored in a number of other theoretical works discussing corporate social

responsibility.

It is notable that, though the document is very skeptical of regulators, the inaugural work on credence goods endorses a method bearing an eerie similarity to bug bounties:

> "A form of governmental intervention which combines both [government quality standards and distributed monitoring] methods is for the government to prescribe standards for self-rated classes ... which are then enforced by private individuals filing remunerative malpractice suits if the claimed standards are not met. This "bounty hunter" system effectively turns expert buyers into monitors for the less expert customers" [97].

However, this argument assumes that the company will not or cannot adversarially prevent experts from providing analysis of their system. As you will see in Chapter 3, our work on vulnerabilities in voting systems indicates that this concept would likely yield mixed results for software security, as companies may then be rationally incentivized to make it difficult to discover or expose flaws.

## 2.4 Conclusions

Through both new regimes and old, a common (yet infrequently acknowledged) theme is the importance of outside, permissionless monitoring to provide knowledge of breaches of obligations or confirm compliance with standards. It can confirm that the as-deployed or as-run system matches the system that may have gone through prior auditing and inspection. It can alert the public to a need to use legal tools of investigation and discovery. In short, reverse engineering and security analysis is the binding glue in our schemes for software accountability, and when paired with legal responses produces a level of quality control that cannot be matched by either law or technology alone.

We will see in Chapter 3 that looking for these fraudulent labels can help guide us to examine particular markets where security has likely failed, and the impact achieved serves as an existence proof that providing information to the market really can alter

behavior. Chapter 4 demonstrates that the information asymmetry between users and their systems is so fraught that security failures, even when encoded bold-faced in *publicly available standards*, may be so conceptually abstract as to be beyond the average user. However, that same chapter demonstrates how security research can work to realign incentives by altering such a system's performance to obviate this asymmetry. Unfortunately, not all regulation in this space is well-founded to align incentives toward user security, as evident in the debate surrounding law enforcement access to encrypted data as discussed in our final Chapter 5, and so it may fall on security researchers to explain and prevent such failures from occurring.

# Chapter 3

# The (In)Security of Internet Voting

Intuitively, there are a number of reasons one might expect that the systems involved in real-world elections would be very difficult to break. In a practical sense, these systems are inherently important to the continued legitimacy of our democratic institutions, and the state should inherently want to ensure their designs are correct. Political and legal pressure, especially from losing parties, should naturally lead to further intense scrutiny. Even better, the needs and requirements for such systems have been extensively studied — quite literally since the time of Plato — so the design of elections systems should be very well understood. For digital systems, voting has been studied extensively in the cryptographic literature since the early 80's, and served as motivation for some of the original works on multiparty computation and blind signatures.[1]

The goal of this chapter is to demonstrate that, despite all of the natural arguments above, the economic and regulatory environment surrounding voting systems has led us to a place where voting systems (particularly those used in Internet voting in the U.S.) are not secure and not private. To do so, we present a security analysis of internet voting systems used in U.S. federal elections.

We begin in Section 3.1 with an overview of useful background on Internet voting systems in the U.S., the legal and policy status of such systems (3.1.1), and a tutorial

[1]Indeed, we will see in Section 3.1.2 that the requirements for voting systems are often subtle and surprisingly hard to get right.

on the related work in security and cryptography of voting systems (Section 3.1.2). We then describe our methodology (Section 3.2), and provide an analysis for each voting system (Voatz in Section 3.3 and OmniBallot in Section 3.4). Finally, Section 3.5 concludes with a discussion summarizing the findings of these analyses and paths forward.

Before we begin, a disclaimer. This chapter includes discussion of vulnerabilities in voting systems deployed in real, high-stakes federal elections in the U.S. and abroad. We would like to emphasize that nothing in this chapter should be taken to indicate that any previous election was called incorrectly, and the author stands by the letter he, along with 58 other security researchers, signed affirming that there has been no convincing evidence of result-altering malfeasance in the 2020 U.S. federal election [7].

## 3.1 Background

The U.S. 2020 elections were, at a minimum, eventful. Beyond the drama endemic to a particularly contentious and high-stakes U.S. presidential election, those in election administration were faced with a unique set of challenges that had not been seen since the 1910's — the COVID-19 pandemic forced states to prepare for an election in which many voters may not be able to cast their ballot safely in person.

As a result, many jurisdictions turned to forms of online ballot delivery and return to facilitate remote participation. One avenue for doing so was a Boston-based startup's mobile app called "Voatz," and another was Democracy Live's more established OmniBallot system, a web-based platform that had been previously used for blank ballot delivery, ballot marking, and online voting.

Both systems had historically enjoyed nontrivial deployment in high-stakes elections, and were slated to have their use expanded. OmniBallot, for example, had long been used to let voters print ballots that will be returned through the mail, but in early 2020, for the first time, three states announced plans for large classes of voters to use it to return their ballots online. New Jersey made the online voting option available to voters with disabilities, calling the move "a pilot for if we need to use it more

broadly in the future" [140]. West Virginia allowed not only the disabled but also military voters and residents overseas to vote online using OmniBallot [219] in the 2020 presidential race, and Voatz in a number of previous elections. Most significantly, Delaware [135] offered OmniBallot online voting during the presidential primary to all voters who were sick or were self-quarantining or social distancing to avoid exposure to SARS-CoV-2—practically the entire state [135, 79].

Unfortunately, the consensus of experts in election security and national security is that the risks of online voting are unacceptable. Numerous studies of Internet voting systems used or slated for use in real elections have uncovered critical security flaws (e.g., [323, 280, 138, 157, 159, 277]). The National Academies of Sciences, Engineering, and Medicine concluded that "no known technology guarantees the secrecy, security, and verifiability of a marked ballot transmitted over the Internet," and that, "[a]t the present time, the Internet (or any network connected to the Internet) should not be used for the return of marked ballots" [224]. In light of Russia's attacks on U.S. election infrastructure during the 2016 presidential election, the Senate Select Committee on Intelligence has recommended that "[s]tates should resist pushes for online voting," including for military voters [300]. In May 2020, the Cybersecurity and Infrastructure Security Agency, Federal Bureau of Investigation, U.S. Election Assistance Commission, and National Institute of Standards and Technology privately warned states that "electronic ballot return technologies are high-risk even with [risk-mitigation] controls in place," and that attacks "could be conducted from anywhere in world, at high volumes, and could compromise ballot confidentiality, ballot integrity, and/or stop ballot availability" [311].

Despite these risks, neither OmniBallot or Voatz had previously been the subject of a public, independent security review,[2] and there was little public documentation about either system's security guarantees.

Increasing voter access is a laudable goal. Voters who are sick, disabled, or stationed overseas sometimes face substantial obstacles to participation, and the coronavirus

[2]Both Democracy Live and Voatz made claims that audits have been conducted by the National Cybersecurity Center (a private entity) [225] and ShiftState Security [102], though only high-level summaries of these audits had been made public for either.

pandemic brought into sharp relief the need to provide resilient voting alternatives to everyone. However, elections also face substantial risks from attackers—risks that are magnified when delivering or returning ballot online, and the aftermath of the 2020 presidential election highlights the need for election administration to avoid even the *perception* of vulnerability.

### 3.1.1 Legal, Economic, and Policy Background

While the introduction of Internet voting in the U.S. is relatively new, the history surrounding electronic only voting is not. In the wake of counting errors, recount discrepancies, and uninterpretable ballots wreaking havoc during the 2000 U.S. Presidential race, Congress passed the Help America Vote Act (HAVA) [229], a bill targeted toward helping states move away from outdated and problematic punchcard-based systems. The Election Assistance Commission (EAC), a new executive agency created by HAVA, was charged with distributing these funds, and has since provided over $3.3 billion to various states to help improve election infrastructure [117].

Unfortunately, HAVA lacked stringent guidelines on what replacement systems were allowed to be purchased. As a result, many states acquired unvetted electronic-only voting machines, known as Direct-Recording Electronic (DRE) systems. Numerous studies have since shown DRE systems to be extremely vulnerable to a wide range of attacks, allowing adversaries to surreptitiously change the outcome of an election [125, 331, 56, 71, 187].

In 2020, we began to witnessing similar developments in response to Russia's interference in the 2016 U.S. Presidential election. Bills had been introduced in both the U.S. Senate [186] and House [291] that aim to provide funding to revamp election infrastructure. At the same time, there had been renewed interest in cryptography due to advances in accountable and transparent systems such as the blockchain [221], and the proliferation of mobile devices.

The result was increased speculation surrounding the potential use of personal devices to allow voters to cast their ballots over the Internet. At the time we began the work presented in this chapter, there were at least four companies attempting to

offer internet voting solutions for high-stakes elections [211], and one 2020 Democratic presidential candidate included voting from a mobile device via the blockchain as a part of his policy plank [21].

### U.S. Elections System Market Incentivizes Poor Security

Elections administration in the U.S. is fractured, with each state maintaining its own set of regulations, organizational structures, and budgeting. A local elections official in one state may, for example, have the power to select what voting systems they use, where in another this may be mandated by the state.

Unfortunately, this multiplicitous approach exacerbates the already difficult problems associated with information asymmetry in computing. Elections officials are tasked with making difficult purchasing decisions about the systems they use, yet officials are often short-staffed, underfunded, and lack the ability to perform an independent evaluation themselves.

Worse, nation-wide regulatory attempts at improving security have focused mainly on providing very minimal usability and security standards rather than incentivizing third-party analysis, improvement, and transparency. Likely the most relevant rules for regulating elections systems are those codified by the EAC in the Voluntary Voting System Guidelines (VVSG) [299], which considers "transparency" as a question of documentation, rather than independent security analysis. And, though this *could* be seen as a use of labeling to show adherence to a set of substantive minimum standards, systems are "certified" as adhering to the standards in the VVSG through private entities that are paid by the software vendors themselves, and it is unclear if there is a process in place to disincentivize cheating by the auditors (indeed, as we will see later in this chapter, it is clear that the auditors do not always behave well).

In any case, the VVSG itself explicitly *does not consider* Internet voting systems, but was instead designed for in-person ballot marking devices to be used in a physical polling place. Barring state-level requirements (which are relatively rare), the only U.S. regulation on the security of voting systems simply does not apply to these systems.

In sum, the market for Internet voting systems in the U.S. suffers from many of

the hallmark symptoms of concern to economists that we discussed in Chapter 2. The technical obscurity of computer security as a credence quality, a set of purchasers that cannot effectively evaluate vendors' claims, a complete lack of labeling and minimum standards, and a failure of regulation to require transparency and third party analysis all lead us to conclude that a security analysis of such systems should be fruitful.

### 3.1.2 Background on Cryptography and Voting

Voting as a research subject in both applied vulnerability discovery and in cryptography is not new, indeed, election administration has been used as a motivation for novel cryptographic primitives as early as the 1970's. As a result, the security community has provided a number of formal requirements on voting systems, largely codifying the results from various social science studies.

**Evidence-based elections:** An evidence-based election holds the property that election officials are not only capable of calling an election correctly, but providing "...the electorate convincing evidence that they did" [281, 27]. This compelling requirement implies both that the election system must be *auditable* (meaning it creates an evidence trail that can be checked to confirm that each relevant part of the system is functioning correctly as intended) and that *any given election* run using that system must be *audited* (meaning that the evidence trail is actually checked in that given instance).[3] *Auditability* alone is insufficient, and must be accompanied by *auditing* to be effective: auditability without auditing is like collecting receipts so one can check a credit card bill, then never checking the receipts against the bill. In short (paraphrasing [281]), auditability + auditing ⇒ evidence-based election.

Next, we highlight five *minimal* — necessary but insufficient — requirements for secure elections in an evidence-based framework: (1) ballot secrecy; (2) software independence; (3) voter-verifiable ballots; (4) contestability; and (5) auditing.

[3]The term "audit" in the elections context is often used to refer to *post-election audits*, a particular type of auditing that checks some subset of the ballots after the main count is performed, in order to check that those ballots are consistent with the claimed outcome. Here, the term "audit" is used in its more general sense, encompassing checking or verifying the correct functioning of system components at any point during the election process.

**1. The secret ballot** Ballot secrecy is essential to combat voter corruption and coercion. As the U.S. Supreme Court has put it, "a widespread and time-tested consensus demonstrates that [ballot secrecy] is necessary in order to serve... compelling interests in preventing voter intimidation and election fraud." [70] Protecting ballot secrecy provides a strong and simple protection against coercion and vote selling: if you cannot be sure how anyone else voted, this removes your incentive to pay them or threaten them to vote the way you'd like.[4] Indeed, election law scholars have noted that "[b]ribery of voters was far and away the greatest impediment to the integrity of elections before the introduction of the secret ballot, a fact well known not only to historians but to readers of great 19th century fiction"[5] [204].

In a formal, cryptographic sense, secret ballot voting systems need to ensure that:

1. No voter is able to prove their selections (*Receipt-Freeness*),
2. that no voter's choices can be surreptitiously released or inferred (*Privacy*), and
3. that a voter cannot cooperate with a coercer to prove the way they voted (*Coercion Resistance*).

These properties are required to provide an election free from undue influence: if a voter is able to prove the way they voted, they can sell their vote, and if a voter's preferences are leaked or forced to be revealed, they may suffer harassment and coercion [51, 100].

**2. Software independence** *Software independence* [258, 260] is the property that *an undetected change or error in a system's software cannot cause an undetectable change in the election outcome.* Software independence is a key property to ensure *auditability* of the casting, collecting, tallying components of election systems. And even beyond ensuring that any errors that occur are detectable, software independence also reduces the likelihood of large-scale errors or attacks occurring in the first place:

[4] After taking your payment, they would have no incentive to follow your instructions; and in the case of coercion, there would be no way to credibly follow up on the threat.

[5] They reference Charles Dickens' *Bleak House*, George Eliot's *Felix Holt, Radical*, and Anthony Trollope's *Doctor Thorne*.

software-based systems are much more susceptible to *scalable* failure than non-software-based systems. For example, a remote programmer changing a line of code could in principle change millions of electronic ballots in milliseconds, whereas changing millions of paper ballots requires physical access and one-by-one handling.

Software independence does not require systems to not use software at all: rather, it means that the work of any software-based piece of the system (including auditing components) be *checkable, in principle*, using non-software-based means.[6] For example, a system for ballot casting, collection, and tallying would need to produce an evidence trail with an associated verification procedure to check that the system (1) recorded votes as intended, (2) collected them as recorded, and (3) counted them as collected, in any given execution. The basic definition of software independence leaves open *by whom* errors should be detectable: the appropriate answer to this question depends on the context, but using the tripartite framework just mentioned, *individual voters* should be able to detect errors in (1) and (2), and *anyone* should be able to detect errors in (3). Who can verify (1) and (2) is constrained by the ballot secrecy requirement from above.

Without software independence, an undetected error in a piece of code could cause an undetected or unconfirmable error in the election outcome — our state of the art is far from achieving error-free code. Democracy — and the consent of the governed — cannot be contingent on whether some uncheckable software correctly recorded voters' choices.

**3. Voter-verifiable ballots** Even before ballot casting, a voter composing a ballot must be able to verify for herself that her prepared ballot reflects her intended choices. Paper ballots inherently enable simple verification that ballots are recorded as intended: a property that is challenging for electronic-ballot systems to achieve. "With a hand-

[6]In practice, the *practicality* of running the verification procedure with minimal or no dependence on software is an important issue — and one unaddressed by software independence, which just represents a minimum threshold of auditability. To facilitate auditing with little to no realistic reliance on software, it is strongly preferable to have systems that *either* have verification procedures simple enough for people to execute without using software at all *or* have open-source and publicly documented verification procedures, which voters and/or the public can perform using their own software.

marked paper ballot, the marks on the ballot necessarily reflect what the voter did, and we can have reasonable assurance that the human-readable mark on the ballot is for the candidate actually intended by the voter" [27]. A voter looking at their completed paper ballot can directly see whether their intended choices are marked (and, in principle, detect any mistakes they made).

**4. Contestability** Software independence alone leaves another question unresolved: *when an error is detected, can the one who detected it convince others that an error indeed occurred?* Some types of errors may be *publicly* detectable, rendering the second question moot (since then anyone can run the verification procedure for themselves). However, certain verification procedures may be non-public: e.g., certain errors related to a given voter's vote might be detectable only by that specific voter. A *contestable* voting system is one that provides *publicly verifiable evidence* that the election outcome is untrustworthy, whenever an error is detected [26].[7]

**5. Auditing** As already mentioned, in addition to being *auditable* (which, for casting-and-tallying systems, corresponds to software independence and contestability), elections should be *audited*. Auditing checks that the evidence is trustworthy and, for casting-and-tallying systems, consistent with the announced election outcome. Both auditability and auditing are necessary for evidence-based elections. Such auditing should include compliance audits and risk-limiting audits [281]. Furthermore, "[t]he detection of any software misbehavior does not need to be perfect; it only needs to happen with sufficiently high probability" [258].

Election equipment may fail. The system must be designed not only to prevent failures, but also to ensure timely detection of failures when they occur: the public has a right to know about failures in the election process. See [27] for a more in-depth discussion of security requirements in evidence-based elections.

[7] It would be even better if the correct outcome could be recovered in case of a dispute; detectability is just a minimum requirement. A system that is software independent and also enables this sort of recovery (based on the evidence trail produced during the election) is called *strongly software independent* [26].

### End-to-End Verifiability

Computer scientists have been working for more than 30 years to develop principled techniques for secure remote voting [47], though how to meet all of the criteria described in the previous section, while also providing a usable system is still an ongoing research effort [51].

These protocols use an approach called "End-to-End Verifiability" (E2E-V), which have the property that voters receive proof that their selections have been included, unmodified, in the final tallying of all collected ballots, *without the need to trust any separate authority to do so* [51]. There have been research prototypes developed that provide such guarantees while maintaining coercion resistance, privacy, and receipt freeness using techniques such as visual cryptography, homomorphic cryptography, invisible ink, and mixnets [82, 265, 41, 84].

Unfortunately, no system has yet been able to adequately provide the same guarantees that voters currently enjoy through in-person poll booths. For example, while the prototype E2E-V protocol Helios [8] does not require the voter to trust a particular client device or the official election software or servers to tally the vote correctly, but fails on privacy and coercion resistance. These technologies are promising — both for remote voting and as an added layer of protection for traditional voting [217] — but they are also complex and difficult to implement correctly [157]. For this reason, although experts hold that E2E-V should be a requirement for any Internet voting system, they simultaneously caution that "no Internet voting system of any kind should be used for public elections before end-to-end verifiable *in-person* voting systems have been widely deployed and experience has been gained from their use" [114].

## 3.2 Experimental Methodology

Researchers have conducted numerous independent analyses of electronic voting systems by acquiring voting equipment, reverse engineering it, and testing it in a controlled environment (see [158] and references therein). Safely testing an *online* voting system is more challenging. Such systems necessarily have server-side components that (unless

source code is available) cannot be replicated in the lab. Accessing non-public server functionality might raise legal issues and would be ethically problematic if it risked unintentionally disrupting real elections [261].

To avoid these issues, we constrained our analysis to publicly available portions of each system. Following similar methodology to Halderman and Teague [159], we obtained the client-side software for each system, as available to any member of the public, reverse-engineered it, and implemented our own compatible server in order to drive the client without interacting with the real voting system. This approach limits our ability to identify vulnerabilities in either system's *server-side* code and infrastructure—an important task for future work—but we were able to learn many details about the platform's design and functionality.

## 3.3 Voatz

To gain a better understanding of Voatz's infrastructure, we began by decompiling the most recent version of their Android[8] application as found on the Google Play Store as of January 1, 2020 and iteratively re-implemented a minimal server that performs election processes as visible from the app itself. This included interactions involved in device registration, voter identification, and vote casting. We used two devices for our dynamic analysis and development: a Voatz-supported Pixel 2 XL running Android 9, and a Voatz-unsupported Xiaomi Mi 4i running the Lineage OS with Android 8, both jailbroken with the Magisk framework [206].

In order to redirect control to our own server, we were forced to make some small changes to the application's control flow. To reduce threats to validity, we limited these modifications to the minimum necessary in order to redirect all network communication. We:

1. Disabled certificate pinning and replaced all external connections to our own servers;

[8] We did no analysis on and make no claims about Voatz's iOS app.

2. Disabled the application's built-in malware and jailbreak detection. Details are available in 3.3.3; and,

3. Removed additional encryption between the device and all still active third parties, re-targeted all communication from these services to our own server, and reimplemented the necessary parts of their protocols as well.

While all of this could have been accomplished by statically modifying the program's code, we instead opted to dynamically modify or "hook" relevant parts of the code at runtime using an Android modding framework. Modifications therefore required no changes to the application code itself, only to code running on our test devices, allowing for rapid development and transparency about what was modified at each stage of our analysis.

Despite this lengthy description, our codebase is relatively simple. The on-device hooking code consists of roughly 500 lines of Java that leverages the Xposed Framework, a series of hooking libraries that are well supported and popular in the Android modding community. Our server implementation is roughly 1200 lines of code written in Python using the Flask web framework.

### 3.3.1 Voatz's Security Claims

Although there is no public, formal description of their system, Voatz does make a number of claims about their system's security properties via their FAQ [308].

**Immutability via a permissioned blockchain:** Voatz claims that once a vote has been submitted, Voatz uses "...blockchain technology to ensure that...votes are verified and immutably stored on multiple, geographically diverse verifying servers." The FAQ goes into further detail, discussing the provision of tokens for each ballot measure and candidate.

**End-to-End vote encryption:** Voatz makes multiple references to votes themselves being encrypted "end to end." To the authors' knowledge, there is no formal definition of "end to end vote encryption;" for example, it is unclear where the "ends" of an end

to end encrypted voting scheme are. It is worth noting that there exist homomorphic cryptography schemes that tally votes over the vote ciphertexts, so that one need only decrypt an aggregate vote, maintaining individual voter privacy [46], but it is unclear from the FAQ if this is what Voatz intends.

**Voter anonymity:** Voatz claims that "the identity of the voter is doubly anonymized" by the smartphone and the blockchain, and that, "Once submitted, all information is anonymized, routed via a 'mixnet' and posted to the blockchain."

**Device compromise detection:** Voatz claims to use multiple methods to detect if a device has been jailbroken or contains malware, and that "The Voatz app does not permit a voter to vote if the operating system has been compromised."

**Voter Verified Audit Trail:** Voatz claims that voters receive a cryptographically-signed digital receipt of their ballot after their vote has been submitted. The guarantees of such a receipt are unclear, although, perhaps this is meant to provide similar guarantees as E2E-V voting systems.

### Prior Scrutiny of Voatz

While we are the first to publish an in-depth analysis of Voatz, others have raised concerns about their system, security claims, and lack of transparency. Jefferson et al [99] compiled a long list of unanswered questions about Voatz, including the app's use of a third party, Jumio, as an ID verification service. Several writers observed the election processing and audit of the Voatz pilot during the 2019 Denver Municipal elections, and found that the main activity of the audit was to compare a server-generated PDF of a voter's ballot with the blockchain block recording the same [286, 160]. Kevin Beaumont found what appeared to be several Voatz service-related credentials on a public Github account [37], and that the Voatz webserver was running several unpatched services [38]. Voatz responded citing a report from the Qualys SSL checker as evidence of the site's security [208], and later claimed that the insecure server Beaumont identified was an intentionally-insecure "honeypot operation" [314].

As a result of this public scrutiny, in November 2019, U.S. Senator Ron Wyden called on the NSA and DoD to perform an audit of Voatz [263].

### 3.3.2 Voatz's System Design

In this section, we present Voatz's infrastructure as recovered through the methodology presented in Section 3.2. We begin with an overview of the system Section 3.3.2, illustrating the process by which a user's device interacts with the app during all stages of the voting process including Voatz's custom cryptographic protocol Section 3.3.2, user registration and voter verification Section 3.3.2, and vote casting Section 3.3.2. Finally, we discuss all non-protocol device-side defensive measures we discovered Section 3.3.2.

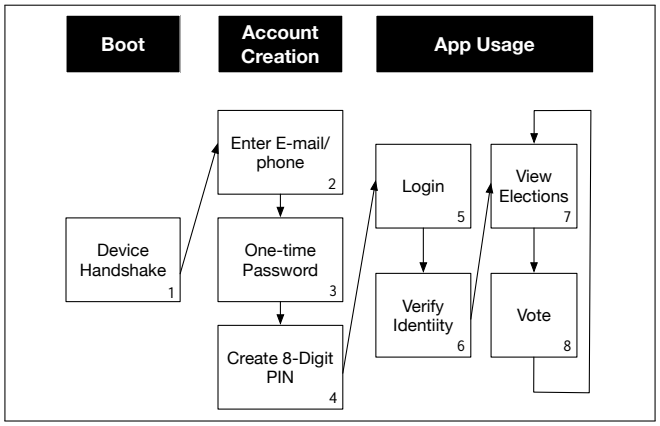



Figure 3-1: **Voatz's workflow as seen from the device.**

#### Process Overview

Figure 3-1 presents a diagram of the steps that occur in-app from login to election voting. They are:

1. The device initiates a handshake with the server, creating a shared key which enables an extra layer of encryption beyond TLS (Box 1). Communication between the device and Voatz server is described in Section 3.3.2.

2. The user creates an account by providing their E-mail address, phone number, and an 8-digit PIN (Boxes 2-4).

3. The user logs in with this PIN (Box 5).

4. The user verifies their identity, using Voatz's integration with a third-party service called Jumio (Box 6). The app requests a scan of the user's photo ID, a recording of their face, and the user's address, and then sends all of this information to Jumio's servers.

5. The user selects from a list of open elections, and then marks and submits their ballot. Depending on the election configuration, Voatz can allow "vote-spoiling,"[9] so this process may be repeated prior to the election closing. (Boxes 7-8)

**Communication** Figure 3-2 shows the communication between components of Voatz and other entities. As we were only able to directly observe communication involving the Voatz app, the rest of this diagram is an attempted reconstruction based on documents released by Voatz [194] and by the Denver Elections Division [129].

The three primary third-party services used by the Voatz app are the identify-verification service Jumio, a crash reporting service Crashlytics, and a device security service Zimperium. Of these, the most significant is Jumio, which Voatz relies on for ID verification, and to which the app sends substantial personal information (see Section 3.3.2).

### Voatz Server Handshake and Protocol

Voatz's server is implemented as a REST application — all communication between Voatz's server and the application occur as a series of JSON-encoded HTTPS GET, PUT, and POST commands. The app's REST server is `voatzapi.nimsim.com`, with `voatz.com` only used for static assets such as images and text. All parts of the protocol leverage the Android OS's built-in TLS stack, and uses certificate pinning to ensure that the incoming certificate is from a particular issuing Certificate Authority.

Next, *on top of TLS*, the system performs a "device handshake" with the following steps:

[9] Vote spoiling refers to casting a new vote that invalidates all previously cast ballots.

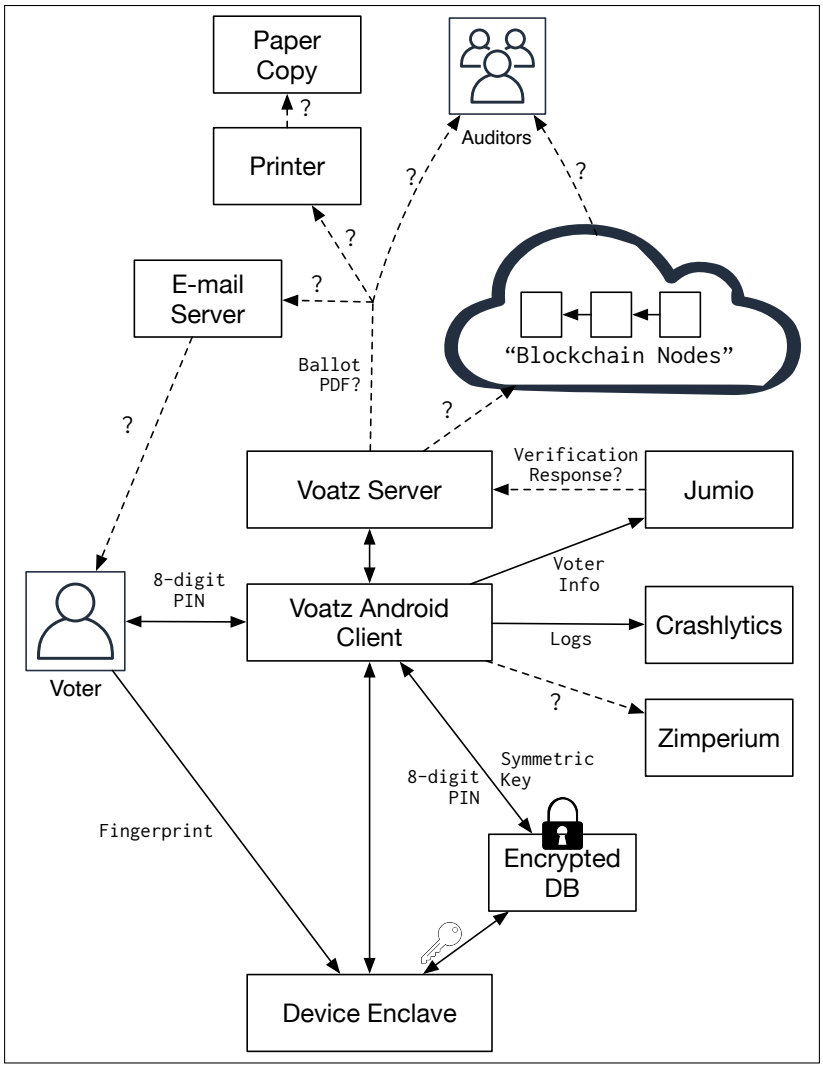



Figure 3-2: Dataflow between Voatz components and external services. Dashed lines are believed to exist but have not been directly observed.

1. The App generates 100 ECDSA SECP256R1 keypairs, and sends the Server all 100 corresponding public keys. The device saves only the 57th keypair $(PK_D, SK_D)$.

2. The Server generates 100 ECDSA SECP256R1 keypairs, selects the 57th $(PK_S, SK_S)$, and performs the rest of an Elliptic Curve Diffie Hellman (ECDH) key exchange to generate a shared secret $(SK_{ecdh})$.

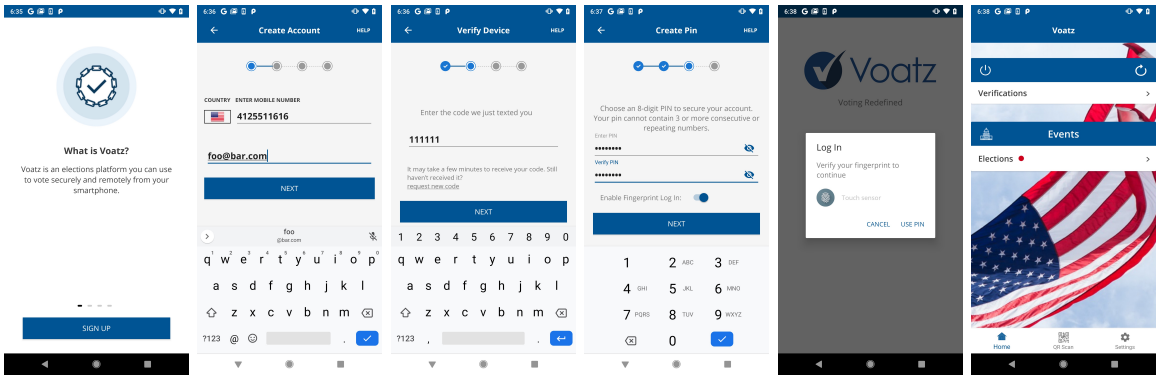
3. The Server generates AES-GCM parameters; a random AES-GCM 256-bit symmetric key $(SK_{aes})$, a random 16-bit nonce $(N)$, and a Tag $(T)$.

4. The Server then sends the device the 100 public keys generated above, including the $PK_S$ as the 57th key and ECDH-Encrypt$(SK_{ecdh}, SK_{aes}||N||T)$

5. Out of the 100 public keys sent by the Server, the App selects the 57th pubkey ($PK_S$), and finishes the ECDH handshake to create the ECDH shared key $SK_{ecdh}$. Finally, it decrypts and parses the AES-GCM parameters$(SK_{aes}, N, T)$.

This handshake is performed every time the app is launched for the first time, and, from this point forward in the app's execution, *every communication* between the App and the Server is encrypted using the standard AES-GCM algorithm by way of $SK_{aes}$, in addition to the encryption provided by TLS. Note that there is no authentication of the ECDSA keys by the app, beyond the encapsulating TLS certificates. This made it very simple to retarget the server — we replaced all required URLs in-app to our own and followed the protocol. Further, this renders the use of the handshake somewhat unclear, as it offers no protection against active MITM attacks over the authentication already provided by TLS.

It also is worth mentioning that all but the 57th keys are abandoned immediately on the device side — both the extraneous secret keys the device generated in the first step and the public keys it receives from the server. We conclude that this 100-key exchange is likely an attempt at obfuscation, rather than serving any useful purpose to the security protocol.




Init Screen | Contact Info | OTP | PIN | First Login | Main Screen

Figure 3-3: The user registration process, connecting to our server reimplementation.

### User Registration & ID Verification

After the app has completed the device handshake, the user can begin the registration process, which can be seen in Figure 3-3. Here the user is asked to submit their email and phone number, and perform a One Time Password operation via SMS. Finally, the user selects an 8-digit PIN number which is then sent to the server, and used extensively in user authentication.

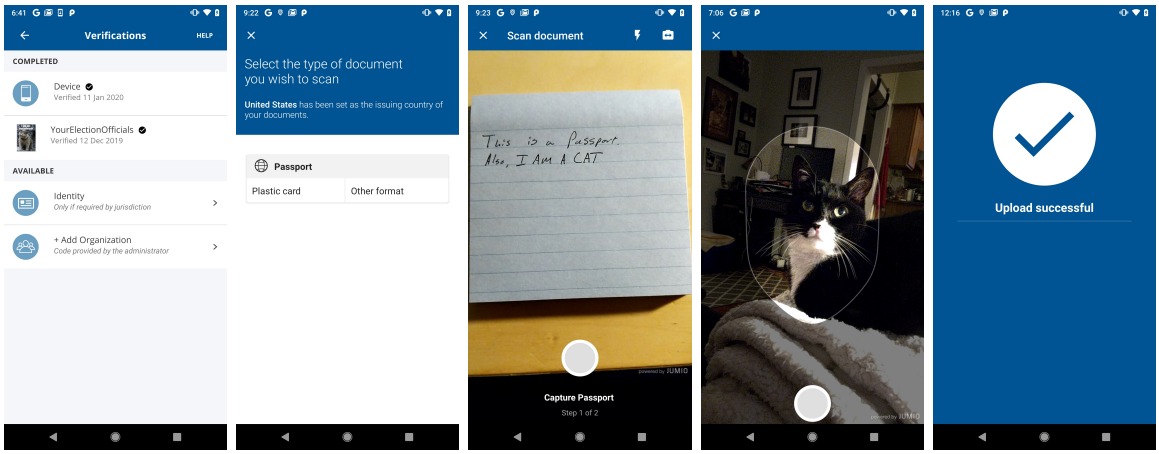
If the user has a fingerprint registered with their device, they are given the option to "enroll" their fingerprint as an alternative authentication mechanism. Effectively, this works by storing the PIN on-disk, encrypted using a key biometrically tied to the user's fingerprint via the Android Keystore.

The Android Keystore is a system service that, if used correctly,[10] will perform various cryptographic operations on behalf of the application, on application-level data, *without* exposing the requisite key material to the application's host memory.[11] Further, when supported by the device's hardware, these device-level keys are stored in the manufacturer's protected hardware, and can be made to require the user to enter in their device password or fingerprint before they are used.

After registration, the user is asked to log in via the PIN (or fingerprint decryption of the PIN). In addition to the PIN, there are four pieces of information sent to the server to authenticate the user at log in: a unique device ID generated via Android's ANDROID_ID system,[12] a customer ID number, a "nextKey" value, and an "auditToken". The nextKey and auditToken are originally received from the API server, are never modified except when updated by the server, and do not appear to be used in any device-side cryptography. How these authentication parameters are stored is explored in Section 3.3.2.

After authentication, the user may still need to provide some proof of identity, which requires visiting the verification menu from the main screen (Figure 3-4a). When the user selects the identity option, the app launches Jumio's sub-activity to select a

[10] In Section 3.3.3, we will see that the app does not use the Keystore correctly.
[11] See Android's Keystore documentation for details [22].
[12] See [23] for more information about Android's local device UUIDs.




(a) ID Verification Status (b) ID Type Selection (c) Picture ID Photo (d) Face "Selfie" (e) Verification Success

Figure 3-4: The voter verification process as seen from our experimental environment.

document type (Figure 3-4b[13]). The user is prompted to take a photo of their ID or Passport (3-4c), and to take a selfie photo (3-4d), after which a dialog prompts the user for their registered voting address (not pictured). The app then uploads data to Jumio's server, including the user's photo, the voter's name, address, and photo ID (3-4e).[14] Finally, after receiving a response from Jumio's server, the app sends a subset of the user's data to Voatz's server as well.

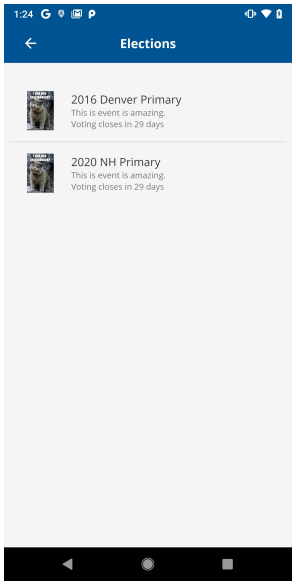
It is worth noting that the small, translucent logo in the bottom right corner of the photos taken during this process (Figures 3-4c, 3-4d) appears to be the only in-app indication to the user that Jumio exists, and the only way a user would be aware that this data is sent to a 3rd party.

### Vote Casting

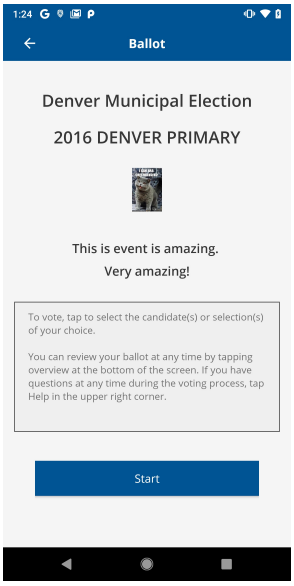
After the user is verified, the app queries the server for configuration data relating to what events the voter is allowed to participate in, activating a menu for the user to select from available events (see Figure 3-5). This configuration data includes all

[13]These options are based on metadata about available documents sent from our mock implementation of Jumio's servers. This screen may differ when the app is connected to the genuine Jumio servers.

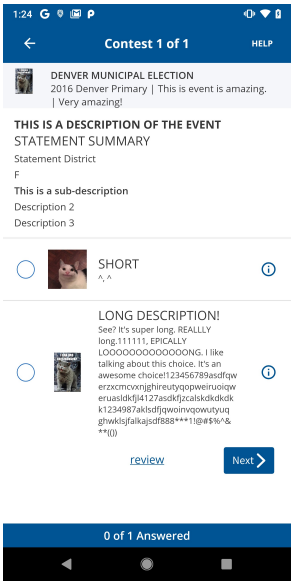
[14]Furthermore, Jumio itself has disclosed that it uses a third party, Facetec, to help analyze the video selfies [284]. As we do not have visibility into their services, we cannot confirm whether or not Jumio actually transmits information to Facetec-controlled servers.



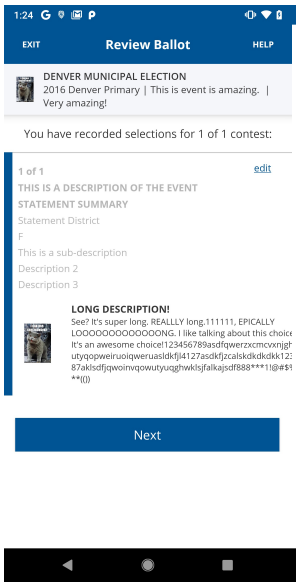

(a) Event Selection



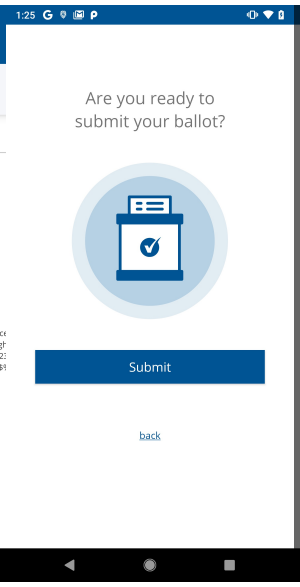

(b) Begin Ballot



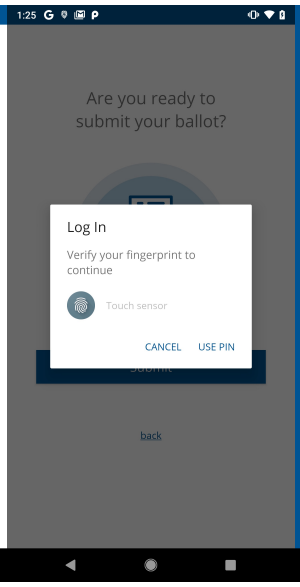

(c) Ballot Question



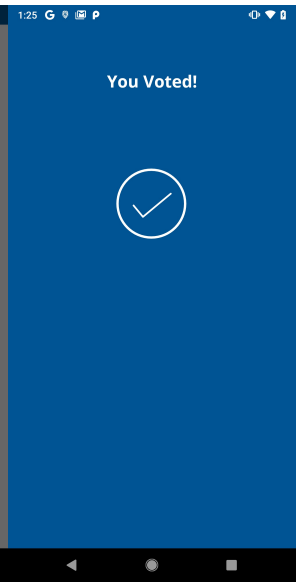

(d) Review Selections




(e) Final Submission




(f) Pin Decryption




(g) Success Screen

Figure 3-5: The voting process as seen in a mock election

events to which the voter has access, those events' ballots, each ballot's particular questions, and the options available for those questions.

The voter begins by selecting an event (3-5a), and is then able to view questions associated with these particular events, select responses (or no response at all, depending on the event configuration), and submit their response to the server. At the point of submission, the user is again asked to decrypt their PIN (3-5e), which is used

as a final authentication mechanism before the ballot is submitted to the server.

It is important to note that the vote is *not* submitted directly to any blockchain-like system, and is instead submitted via this API server. Additionally, although the user is asked to authenticate before submission, beyond the MAC associated with the AES-GCM algorithm and enclosing TLS session, the text of the vote itself is not otherwise signed. The only indication of blockchain-like tokens being submitted or exchanged is the "auditToken," but this string is never altered by the app, and appears to be a single, static value. Figure 3-6 shows the entirety of what is sent to the server, AES-GCM encrypted, after a user submits their vote.

### Device-Side Defensive Measures

In the process of performing our analysis we discovered that Voatz employs a number of obfuscation techniques, leverages a third party virus scanning service, and uses an on-device encrypted database to protect locally stored sensitive data.

**On-disk encrypted database:** After the registration has been completed, the user's login credentials (the nextKey, auditToken, and customer ID number), as well as the voter's entire vote history, are locally stored in an encrypted database using the Realm database framework [254]. When Voatz's app attempts to query the database, the Keystore asks the user to authenticate via a fingerprint or PIN (see Figure 3-5f), before performing the required operations.

The key for the database is linked directly to the user's PIN; specifically, the system runs PBKDF2 with SHA1 over the PIN to generate the key. Recall that this allows the system to use a fingerprint as an alternative method of decrypting the database — At log in, the app can authenticate via the fingerprint to decrypt the PIN, or use the PIN directly to decrypt the database and gain access to the rest of the app.

**Third-party Malware Detection (Zimperium):** Voatz leverages a third-party antivirus solution called Zimperium. At initialization time, the Voatz app loads Zimperium's code as a separate service and registers a series of callbacks that will

```
{
"voteData": [
    {
      "summary": "Best cat?",
      "questionId": "1",
      "isRCVFlag": false,
      "isRCV": false,
      "description1": "bogus desc",
      "statements": [
        {
          "summary": "Statement Summary",
          "statementId": "Statement ID",
          "description3": "Description 3",
          "choices": [
            {
              "choiceDetails": {
                "imageUrl": "https://bit.ly/36DJbC4",
                "webUrl": "https://bit.ly/36DJbC4"
              },
              "choiceId": "1",
              "description1": "^",
              "nonSelectable": false,
              "description2": "^",
              "description": "Short"
            }
          ],
          "description1": "This is a sub-description",
          "description": "This is a description of the event",
          "maxSelect": "1",
          "gender": "F",
          "description2": "Description 2",
          "district": "Statement District"
        }
      ],
      "description3": "bogus desc",
      "description2": "bogus desc",
      "description": "bogus desc"
    }
  ],
  "auditToken": "SomeAuditTokenValue",
  "controlNumber": "1",
  "customerId": 267732387,
  "eventId": 1
}
```


Figure 3-6: An example plaintext payload for a vote submission in a synthetic election.

alert the API Server if Zimperium detects a threat. This message includes the details of the threat, the user ID, and device ID, and the IP address of the offending device.

Zimperium's scans include (but are not limited to) known exploit proofs of concept, known malware, and indicators that the user has installed known superuser tools indicative of a rooted / jailbroken device. Additionally, Zimperium will trigger callbacks if the user appears to have enabled Android's local debugging features such as remote `adb` debugging.

**Partial Code Obfuscation and Packing:** Without the developer taking extra precautions, Android apps may be readily unpacked and decompiled to near the original source via easy to use tools such as APKTool [25] and JADX [274]. However, much of the Voatz app is obfuscated using a packer that presents several barriers to analysis.

First, many of the classes and function names were renamed to random Unicode strings. Beyond making the resulting decompilation more difficult to read, this obfuscation also caused APKTool to crash, while JADX successfully completed decompilation, but left many of the resource files (including application strings and images) unreadable. Voatz's app also contained a few zip files that appear to perform a zip bomb attack [127], which defeats some implementations of `unzip`. Finally, all included 3rd-party native libraries for ARM failed to open in our version of IDA, although it is unclear if this was an active defensive measure as they were successfully disassembled using Ghidra.

We were able to defeat the obfuscation by intensive manual analysis and, in some cases, were aided in recovering the original variable names by the app itself. First, the app uses many libraries which internally depend on Java reflection, rendering the obfuscator unable to rename any classes or methods referenced in this way. Second, the app and some of its libraries are written in Kotlin. While some Kotlin idioms do not decompile easily to Java, the use of Kotlin overall aids reverse-engineering — the Kotlin compiler inserts many runtime checks into the code, each including a string with an error message to display in case of failure. The class, function, and variable

names are often stored in these strings.

**String Obfuscation:** To further complicate static analysis, the strings that control cryptographic parameters of the device handshake (e.g. "AES-GCM") are obfuscated with an XOR-based scheme and then automatically deobfuscated at runtime. As the strings hidden in this way include error messages generated by the Kotlin compiler, this appears to be the result of an automated tool that had been enabled for only these particular methods.

#### Unconfirmed Portions of the Process

As we lack access to Voatz's servers and deliberately avoided any interaction with them, there are unfortunately a few instances where we are unable to confirm how certain third-party actors in the system behave.

**Zimperium execution confirmation:** Zimperium may communicate back to its own servers confirming that the service is running, and then communicate if Zimperium is active directly to Voatz. To the best of our knowledge, there is no public documentation that suggests this is how Zimperium works, and we find no indication from the callbacks associated with Zimperium that this is occurring. Further, any such mechanism could be similarly subverted with the techniques applied in Section 3.3.3, though likely with more effort.

**Jumio voter confirmation:** Jumio's documentation discusses at length the optional ability to communicate with Jumio's servers for out-of-band verification of a user. Since this is well a documented feature of the system, we assume that Voatz's API server receives confirmation directly from Jumio's servers for ID verification.

**Ballot Receipts and the Blockchain:** According to a Voatz whitepaper, votes are recorded on a 32-node permissioned blockchain spread across multiple Amazon AWS and Microsoft Azure datacenters [129]. Footage of the audit of the 2019 Denver Municipal elections shows that the auditing process consists of manually inspecting

blockchain blocks indicating transactions, obtaining several fields including a hash of the voter's choices. The auditor then manually compares the hash via a lookup table to a PDF displaying the voter's choices. These PDFs are allegedly also printed out by the election authority as a paper record, and are redacted versions of the receipt E-mailed to voters. While we know that, in the Denver election, many voters manually replied to indicate that they received a receipt, there is no evidence that Voatz can automatically verify receipt delivery [160].

In our exploration of the code, we find no indication that the app receives or validates any record that has been authenticated to, or stored in, any form of a blockchain. We further found no reference to hash chains, transparency logs, or other cryptographic proofs of inclusion. We conclude that any use of a blockchain by Voatz likely takes place purely on the backend, or in the receipt stage via the use of some other mechanism.

The only references to voter receipts in-app come from a dialog that requests a passcode from the server, and an (apparently unimplemented) QR code reader. The text of the voter receipt dialog appears to confirm that ballot receipts are indeed sent to the voter via email, and encrypted with the server-provided password (see Figure 3-7). Voatz's QR code reader has functional code for an out-of-band method of receiving organization IDs, which allows the voter to participate in particular events, and a largely unimplemented stub for verifying a vote — attempting to scan a QR code that would start the process of vote verification will result in the "not yet supported" message presented in Figure 3-7.

### 3.3.3 Voatz Analysis and Attacks

In this section, we explore various attacks assuming the role of an adversary that has control over particular parts of the election system. This includes three adversaries with various levels of access to individual parts of the overall infrastructure:

1. An attacker that has control of a user's device,

2. An attacker that has control over Voatz's API server, and

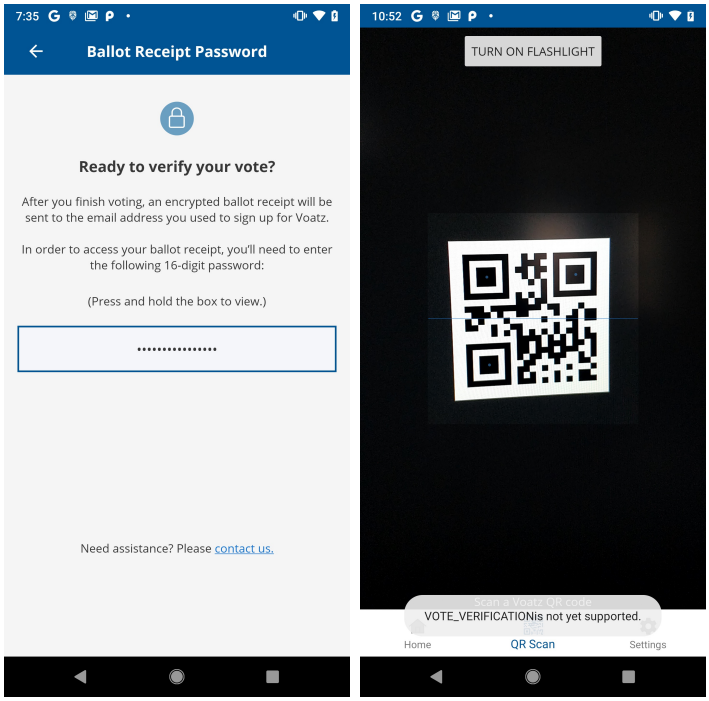



Figure 3-7: Left: the password request screen. Right: the QR code capture screen; note the popup indicating that the VOTE_VERIFICATION QR code type is unimplemented.

3. A network adversary that can intercept network activity between voter's device and the API server, but has no further access.

We believe these adversaries to be credible given the high-stakes nature of the elections in which Voatz is intended to be used, and the resources of the associated attackers. Gaining root control of a user's device can happen through any number of means requiring various levels of skill — via malware, an intimate partner or spouse, as part of a border crossing, etc. Network adversaries could come in similarly many forms, including those that exploit a user's home router (which are notoriously insecure [145, 144]), the unencrypted coffee shop wifi a user attempts to vote from, or the user's ISP.

Including Voatz's API server in this analysis is useful for a number of reasons. While accessing Voatz's server may be more difficult than the user's device and/or the network infrastructure between the server and the user, if the use of Voatz were to be raised to the point that their userbase may alter the outcome of an election, it is not impossible for them to be the target of nation-states, at which point, it is also not outside of the realm of possibility that intelligence agencies would expend considerable

resources, leveraging undisclosed 0-day vulnerabilities, espionage, coercion, or physical attacks, to gain access to crucial systems or key material. Further, a key promise of the blockchain is that it provides an environment where the voter and election authority may trust the system, rather than Voatz, that the election was conducted correctly.

**Assumptions & Threats to Validity** As we lack concrete implementation details about the server infrastructure or backend, we cannot make assumptions about what Voatz logs to their blockchain, the operational security of their servers, blockchain, or cryptographic keys used.

To limit risks to validity, our analysis will make no assumptions about the state of the server beyond what we can glean from the app itself, and we will assume that all interactions, including all cryptographic activities as seen from the device in Section 3.3.2, are logged to the blockchain, and that these blockchain records are secure, monitored, and immutable. This includes all ciphertexts in the protocol, as well as any randomness used in the algorithms.

Note that this is an optimistic analysis of the use of the blockchain in this system. It is unlikely that every interaction is stored via the blockchain, and Voatz's documentation of the West Virginia election indicates that the verifying servers are split equally between Amazon AWS and Microsoft's Azure — indicating that their scheme is vulnerable to Microsoft or Amazon surreptitiously adding resources and executing a 51% attack, or performing a selfish mining attack that requires only 1/3 of the compute power [121].

Nonetheless, we focus on what is provable given our limited access to the system, and show that this analysis is sufficient to demonstrate a number of significant attacks.

### Client-Side Attacks

We find that an attacker with root privileges on the device can disable Voatz's host-based protections, and therefore stealthily control the user's vote, expose her private ballot, and exfiltrate the user's PIN and other data used to authenticate to the server.

```
argClass = loadClass("com.zimperium.DetectionCallback");

findAndHookMethod("com.zimperium.ZDetection", loader,
  "addDetectionCallback", argClass, new XC_MethodHook() {
    void beforeHookedMethod(MethodHookParam p) {
      p.setResult(null); // prevents method from running
    }
});
```

Figure 3-8: Simplified code to disable the Zimperium security SDK.

**Defeating Host-based Malware Detection:** The Zimperium SDK included within Voatz is set to detect debugging and other attempts to modify the app, and to collect intelligence on any malware it finds. By default, it would have detected our security analysis, prevented the app from running normally, and alerted the API server of our actions.

As mentioned in Section 3.3.2, Zimperium communicates with the Voatz app, and ultimately with Voatz's API server, via a set of callbacks initiated when the app loads. Defeating Zimperium was therefore as simple as overriding its entry points to prevent the SDK from executing. The hooking utilities provided by the Xposed Framework allow us to divert control flow with minimal effort — Figure 3-8 shows the code to disable one of its two entry points; in total, disabling Zimperium required four lines of code, and is imperceptible to the user.

We assume that there is no out-of-band communication between Zimperium and Voatz, and find no indication in either Zimperium's documentation or in our analysis of the app that this service exists. If such communication does exist, it would only marginally increase the effort required to defeat it; one would need to hook other parts of Zimperium that perform detection, or communicate with their server directly.

**Full control over the user, on or off device:** Once host-based malware detection has been neutralized, an attacker with root privileges has the ability to completely control the user's actions and view of the app, as well as leak the user's ballot decisions and personal information.

*Stealing User Authentication Data:* Despite being encrypted with keys that leverage

the Android Keystore, the user's PIN and other login information are *not* stored in protected storage, and do pass through the application's memory. Exfiltrating these key pieces of information would allow a remote attacker to impersonate the user to Voatz's servers directly, even off-device.

We find that an attacker with root access to the device can surreptitiously steal the PIN and the rest of Voatz's authentication data. In the process of performing our analysis, we developed a tool that intercepts and logs all communication between the device and the server before it is encrypted with $SK_{aes}$, as well as before data is encrypted and stored in the local database. This allowed us to see, in plaintext, both the user's raw PIN and other authentication data. While our proof of concept stops at logging this information via Android's system debug features (`adb logcat`), it would be trivial to broadcast these requests over the network, modify them, or stop them from occurring at all.

An attacker need not necessarily wait until the user decides to vote — offline attacks against Voatz's scheme are also entirely possible. Recall that the database requires only the user's PIN to unlock, and in no way limits the number of times this PIN might be attempted. Worse, the app artificially limits the PIN to exactly 8 numeric characters, meaning that there are only 100,000,000 possible PINs.[15] A brute force attack can therefore easily rediscover the PIN by repeatedly generating keys and attempting to decrypt the database, recovering the PIN, login information, and vote history of the user all at once.[16]

Such a brute force attack can be performed fairly rapidly. Note that an attacker need not do this on-device, as the encrypted database file can be exported. We implemented a prototype of this attack and confirmed that an attacker can brute-force the key in roughly two days on a 3.1GHz 2017 MacBook Pro. We conclude that such a threat is viable, particularly if the same installation of Voatz will be used across multiple elections.

*Stealth UI Modification Attack:* It is straightforward to modify the app so that it

[15] Voatz also forbids PINs containing 3 consecutive identical digits, which eliminates ~5% of these.

[16] A salt is also required to unlock the database. This is stored on disk, unencrypted, in the app's shared preferences file.

submits any attacker-desired vote, yet presents the same UI as if the app recorded the user's submission. If the election configuration allows vote-spoiling, there is also a variant of this attack previously demonstrated on the Estonian e-voting system: allow the user to vote normally, but change the vote once the user closes the app [280].

Similarly, the attacker could stealthily suppress voter's choices if they select an undesired candidate, but continue to show the verification dialog as if the vote had successfully been cast. To the election authority, this might be indistinguishable from the voter failing to submit a ballot. To the voter, this is indistinguishable from correctly voting, at least until the authority releases voter records for that election.[17]

### Server-Side Attacks

We find that, assuming the optimistic use of the blockchain discussed in the threat model, Voatz's server is still capable of surreptitiously violating user privacy, altering the user's vote, and controlling the outcome of the election.

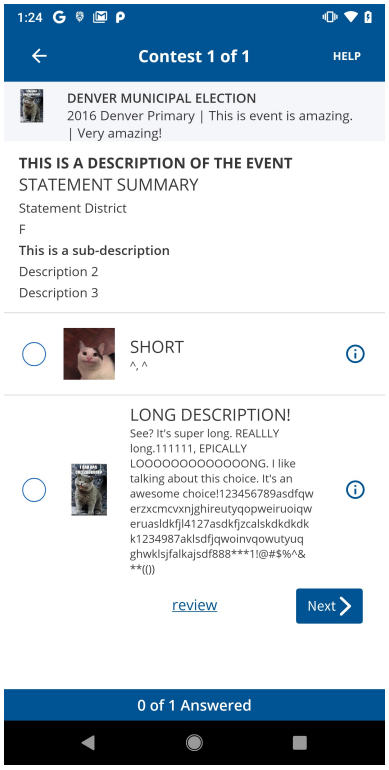
In particular, we find that the protocol discussed in Section 3.3.2 provides no guarantees against the API server actively altering, viewing, or inventing communication from the device; the server can execute an active MITM attack between the user device and whatever blockchain or mixnet mechanism exists on the other end. Note that there is no other cryptographic operation performed between the device and the server at any point other than the AES encryption, including any sort of cryptographic signing by the device or the device's Keystore. If the server performs these cryptographic operations itself — that $SK_{aes}$ is available to the server — it can decrypt the user's ballot before it is submitted to any external log and convincingly re-encrypt any value to be sent to the log.

Even if $SK_{aes}$ is not available to the server — for example, if all cryptographic operations are performed in a Hardware Security Module (HSM) — it must then at least have access to the unencrypted TLS stream, and so it is *still* possible for the server to execute an active MITM attack.

Recall there is no public key authentication performed as a part of the device

[17]For U.S. elections, public records often list which voters participated.

handshake, and there is no proof or verification by the device that these interactions are ever logged on the blockchain. The server can therefore terminate the connection before the HSM and arbitrarily impersonate the user's device by, e.g., replaying the entire device handshake and all future communication back through the HSM to the blockchain.[18] Note that, given these attacks, it is unclear if there exists a scheme in which a receipt can convincingly prove that the correct vote was logged.




(a) Question.

```
Choice = {
   "choiceDetails": {
       "imageUrl":SHORT_IMG,
       "webUrl":SHORT_IMG},
   "choiceId":"1",
   "description":"Short",
   "description1":"^",
   "description2":"^",
   "isWriteIn": False,
   "nonSelectable":False
}
Choice2 = {
   "choiceDetails": {'imageUrl':
       LONG_IMG_URL, 'webUrl' :
       LONG_IMG_URL},
   "choiceId":"2",
   "description":"Long
       Description!",
   "description1":"See? It's super
       long. REALLLY long.111111",
   "description2":"EPICALLY
       LOOOOOOOOOOOOOONG...."
   "isWriteIn": False,
   "nonSelectable":False
}
```

(b) Corresponding JSON.

Figure 3-9: Voting sidechannel attack explained.

**Network Adversary:** We find that an adversary with the ability to view the user's network activity, without access to *any* key material, can still infer how the user voted. Specifically, in this section we demonstrate that the app leaks the length of the plaintext, which can allow an attacker to learn, at minimum, which candidate the

[18]Perhaps this hypothetical HSM also contains the TLS keys required to terminate the connection, and performs all cryptographic operations in the enclave. However, all communication is encrypted with $SK_{aes}$, including those that require queries against databases of users, it is therefore unclear that this is the case, but, even so, the server is capable of performing a number of attacks on the user. See Section 3.3.3.

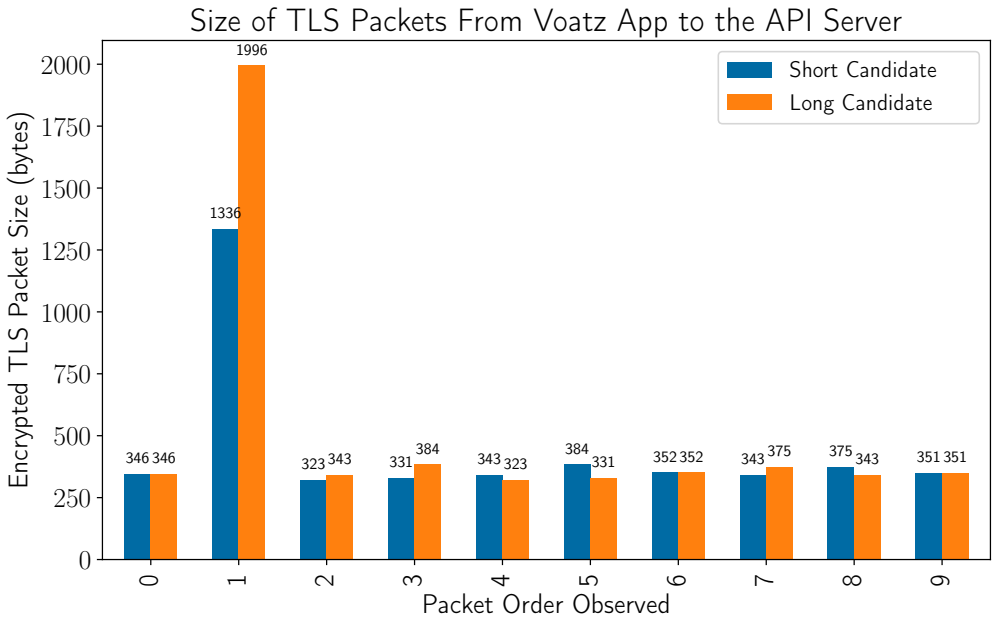



Figure 3-10: TLS encrypted packet lengths immediately after a user submits a vote, in order sent. Note the size of the "short" and "long" candidate in packet 1.

user voted for.

The vulnerability stems from the way in which a ballot is submitted to the server after a user is done selecting their options. As shown in Figure 3-6, the "choices" list in a vote submission contains only the options selected by the user, and includes with that choice *the entirety of the metadata provided by the server about that candidate*. This, in turn, causes the length of the ciphertext to vary widely depending on the choices of the voter.

Figure 3-9b shows the differences in metadata sent to and from the server between the two candidates as displayed in-app in Figure 3-9a. Note that the URLs and other metadata provided are also potentially variable length, and the length of the URL is completely imperceptible to the user.

We verified this vulnerability by setting up a proxy between our app and our API server and recording all communication via `tcpdump`. We then used the app to participate in an election twice, once voting for the "short" candidate and once for the "long" candidate. Figure 3-10 shows the resulting ciphertext sizes in bytes

(specifically, the TLS Application Data field's length per packet) in both runs — in both cases the second packet (packet #1) corresponds to the actual vote submission, where the rest are other miscellaneous protocol queries involved in vote casting and user maintenance. The length of this packet clearly leaks which candidate was selected, is easily distinguishable from other packets in the protocol, and, importantly, its size is unaffected by any parameters that vary by user.[19]

It is worth noting that, ironically, Voatz's additional cryptography exacerbates this vulnerability. In Voatz's implementation, data is gzip-compressed at the application layer prior to being encrypted via TLS, which could have offered some privacy, assuming the compression alone was enough to hide the size differences between plaintexts. Because Voatz encrypts outgoing data *before* the system applies gzip, and compressing an already encrypted payload will not reduce its size, this step is rendered immaterial and the length of the final packet's ciphertext is kept proportional to the size of the plaintext. The result is that (although the figures presented here do intentionally add text to exaggerate the affect for pedagogical purposes), a modest few bytes' difference can be significant enough to determine the voter's preferences.

For this attack to work, we make the following two assumptions:

1. The attacker can learn the ballot options presented (perhaps by themselves voting and gaining access to the JSON representation of the ballot options).

2. The server does not somehow send the ballot options to the device padded to be of equal length.

The first assumption is likely not an issue given the attacks presented in Section 3.3.3. For example, an attacker need only be a registered voter, have previously exploited a registered voter's device and witnessed their ballot options, or otherwise monitored a voter casting a ballot in a particular way and recorded the result.

The second assumption is also a likely to hold, as we find no evidence that the app is defending against this attack — there is no code to remove extraneous symbols

[19]The size of the ciphertext will not vary depending on the user, but may vary minimally depending on the phone's TLS implementation.

or whitespace from ballot questions before they are presented, and other transactions that involve sensitive user information are fully generated device-side and independent of the server (like the user's name, age, and location), and are also not padded. Finally, if this assumption does not hold, a limited version of the attack is still viable: if the user selects *no candidate* and skips the question completely, the device sends the server an empty list.

Note that this sidechannel allows the attacker to detect the voter's intent *before* the ballot arrives at the server. If the attacker is in a position to block packets on their way to the server, (as, for example, an ISP or network owner would), the adversary could intentionally drop this packet and adaptively stop the voter from submitting their ballot. To the user, this would look like a service interruption on Voatz's end, and may degrade the experience enough to stop the voter from casting their ballot at all.

### Other Observations and Weaknesses

**Privacy and geostrategic concerns:** The Voatz app is incredibly privacy invasive. Information sent to Voatz and/or third parties associated with this service include the user's email, physical address, exact birth date, IP address, a current photo of themselves, their device's model and OS version, and preferred language. The app also requests permissions to read the user's GPS upon first login, though we have not identified what exactly the app does with this information. Finally, Voatz makes extensive use of third party code; Voatz includes over 22 libraries provided by 20 different vendors.

One of the reported uses of Voatz's software is overseas military voters, indicating that information leaked about its users could also potentially provide adversaries with information about U.S. military deployments. Note that the voter's IP address alone can carry information about the user's location — so Jumio, Crashlytics, and Zimperium can therefore infer troop deployments.

**Susceptibility to Coercion:** As mentioned in 3.3.2, the app never requires the voter to re-enter their PIN at log-in after registration, and does not appear to show the user if a ballot has been re-voted or spoiled. This indicates that the app leaves users vulnerable to coercion attacks. Consider a voter asleep or otherwise incapacitated. Assuming the attacker has physical access to the device and user, and that the device is unlockable via the user's fingerprint, an attacker would easily have the ability to cast a vote on behalf of the user. This threat model is very relevant in the case of intimate partner abuse [89, 207].

**Vote Receipt Has Unclear Benefit:** From what can be discerned from the available documentation and the app's code, it is very unclear what guarantees Voatz's receipt provides. Outside of the password request feature mentioned in Section 3.3.2, there is no mention of the receipt in the app or its binary, and it does not appear that the app provides any method of verifying that the ballot was counted in the blockchain of record — or, beyond Voatz's documentation, that any such blockchain exists.

It is further unclear if Voatz's system is E2E-V. E2E-V systems in the research literature usually require a voter to visit a polling place and use a paper ballot (e.g. Scantegrity [84] and StarVote [41]), an out-of-band communication before or after the election (see, e.g., code voting [83] and Remotegrity [328]), or a means of performing cryptographic challenges at submission time (see Helios [8]). Assuming that the PDF sent to the user contains no running code, how the system could possibly achieve E2E-V would be difficult to ascertain, and, while Voatz's FAQ appears to tout voter verifiability, it does not explicitly claim to be E2E-V.

In any event, there are significant practical challenges in providing such receipts. In the case that the app *did* present some sort of concrete cryptographic verification without E2E-V, this could allow the user to prove the way they voted — violating the requirements of receipt freeness and coercion resistance. If the receipt arrives as an encrypted PDF, it is unclear how Voatz can prove to the user that the encrypted PDF actually came from Voatz, and, if it is verified in-app, how one would protect

the verification process from the UI modification attacks presented in Section 3.3.3.

Finally, there are significant usability concerns of the receipt that require analysis — What remediation does a user have if the submitted ballot and receipt do not match? How does a user know when to expect a receipt? If the receipt is sent or delayed until post-certification of the election, is there no remediation of a mistake? How does one incentivize voters to perform the challenges required for the verification system to be effective? We further note that many of these questions are rooted in open research problems in the E2E-V space [51], and, without further information, a full analysis of these receipts is not possible.

## 3.4 Democracy Live's OmniBallot

From the user's perspective, OmniBallot is a single-page web app, customized for a particular jurisdiction as an independent website. The app is written using the AngularJS framework and implemented as a combination of static HTML, JavaScript, CSS, and JSON-based configuration files. This code runs in the voter's browser and performs all steps of the voting process via a series of API calls to services controlled by Democracy Live. Below, we explain how we performed our analysis, describe the overall architecture of the platform, and provide details of the web app's operation.

For our analysis, we focused on the instance of OmniBallot deployed in Delaware, which was available at https://ballot.elections.delaware.gov/. As of June 7, 2020, the site used OmniBallot version 9.2.11, which we believe was the most recent version of the system at that time. We began by visiting the site and saving copies of the files that comprise the client. We beautified [201] the minified JavaScript files and ensured that they would not communicate with any live election services by replacing references to `*.omniballot.us` domains with `localhost` and disabling Google's services.

Next, we iteratively reverse-engineered the code to understand each server API call and the format of the expected response, repeating this process until we could complete the voting process using a local stand-in server we created. Finally, we confirmed and extended our reconstruction of the system's operation by inspecting

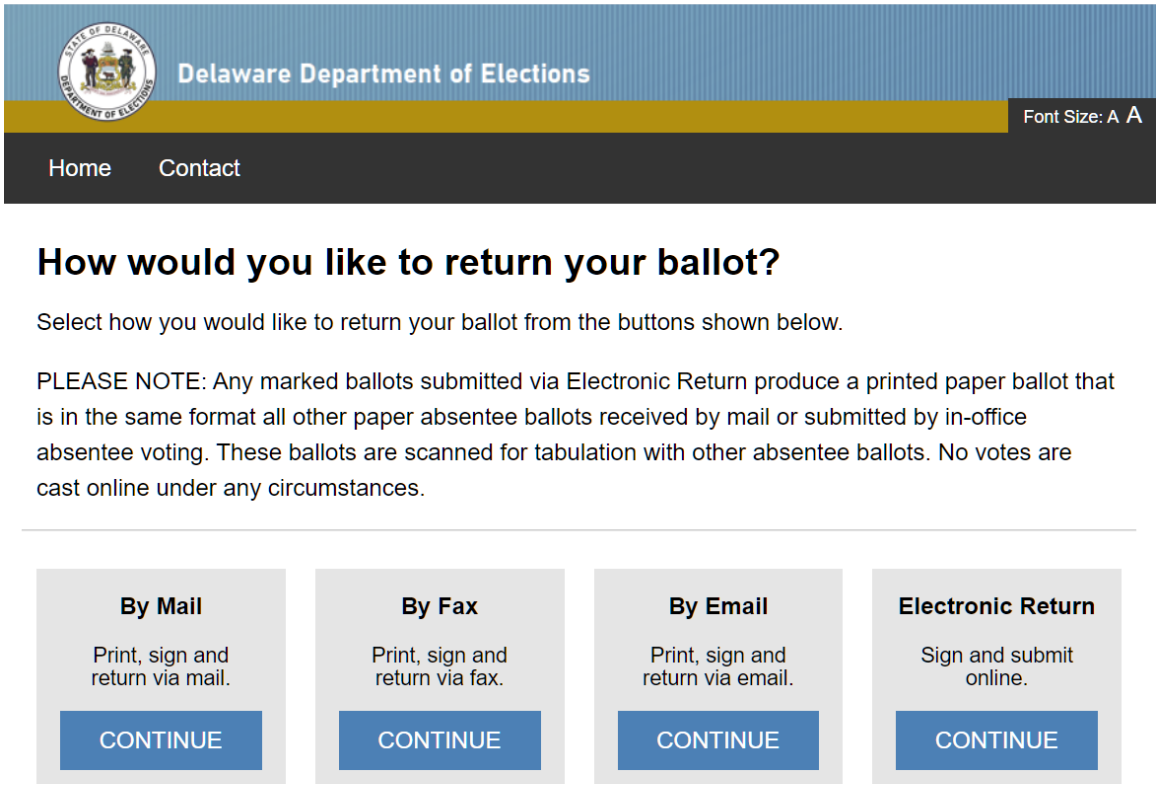



Figure 3-11: **Misleading statements about online voting.** The Delaware app stated that, "No votes are cast online under any circumstances." In fact, both email and electronic return cast the ballot over the Internet. Such mischaracterizations make it harder for voters to understand the risks of their selected return path.

HTTP traces captured by a Delaware voter while using the live system.

### 3.4.1 OmniBallot's Security Claims

**Electronic Return is Somehow Not Internet Voting:** Democracy Live's CEO has repeatedly argued that the online ballot return capability should not be considered Internet voting at all, but rather a "secure portal" or "document storage application" [238]. This claim also pops up in their application and is presented directly to users as a prompt before the user selects their return method (see Figure 3-11).

This appears to be hair splitting over what "casting" a ballot really means, though Democracy Live's interpretation makes relatively little sense. In fact, their use of electronic return completely matches the definition of Internet voting as used by security experts [1] and by the Election Assistance Commission [298]. And, as we

will see in later sections, printing out the ballot after it has been emailed or sent over Democracy Live's system does not substantively change the threats to election integrity.

**Controls via AWS's Tooling:** Available documents give us some visibility into Democracy Live's server-side defenses and internal controls. These controls appear to have either limited or no ability to prevent the attacks described later in this chapter.

The company says that voted ballots are stored immutably in Amazon S3 using AWS Object Lock [103].[20] While Object Lock can only protect files from modification after they are stored, so it cannot prevent attacks that modify the ballot before it is placed in S3. It also cannot protect ballots from modification by insiders at Amazon with internal access to the storage system. Moreover, Democracy Live appears to use Object Lock in "governance mode," which means the protections can be bypassed by the root user or other insider accounts with special permissions [104].

Following a pilot of electronic ballot return during a January 2020 election held by Washington State's King Conservation District, Democracy Live conducted what it called a "post election security audit" in order to "verif[y] the integrity of the [...] election" and "identify potential malfeasance on the part of Democracy Live employees." An unpublished report by the company [104] explains that the "audit" consisted of a review of log entries created by Amazon's AWS CloudTrail log service [17], and it lists ten specific log queries that were performed. We note that these queries did not cover all vectors by which insiders or other attackers could have modified votes. For instance, although the audit included looking for log entries that would occur if an employee logged in under the root account or attempted to remove a restriction on bypassing Object Lock, it apparently did not search for attempts to modify the software downloaded by clients or the software running the `lambda` service. As we have explained, changing either piece of software would be sufficient to allow an attacker to view and alter votes.

[20] Object Lock refers to a configuration of Amazon's S3 storage service that allows the developer to designate certain classes of information unmodifiable for various retention periods and configurations [15].

Such a limited analysis is insufficient to verify the integrity of an election, as it cannot detect the full range of sophisticated threats that public elections face. No matter how comprehensive, server-side logs cannot protect against client-side attacks or attacks conducted through third-party services, since such events would occur outside of Democracy Live's control. Likewise, no level of auditing or procedural controls can eliminate the threat that attackers will introduce malicious functionality into software without detection, and deliberate vulnerabilities can be extremely subtle and difficult to detect (e.g., [85, 126]). Internal audits also provide little assurance against the threat that the employees who conduct them are themselves malicious. Finally, reviewing logs is necessarily retrospective, so, even if a vote-changing attack was uncovered, detection would likely occur only after the election. Since Internet voting lacks voter-verified paper records from which the correct votes could be recovered, officials might be forced to rerun the election.

### 3.4.2 OmniBallot's System Design

Much of what is publicly known about OmniBallot comes from a small number of sources, including an FAQ provided by Democracy Live [103], information posted on various sites for jurisdictions' deployments (e.g., [102]), and press statements by the company. In this section, we provide a more complete picture of the system's operation and adoption, based on our own examination of the software.

#### Modes of Operation

Each jurisdiction's OmniBallot deployment takes the form of a website at a unique URL. The platform is highly configurable, and jurisdictions can customize the available languages, accessibility options, voter lookup and authentication functions, and available features. Most importantly, jurisdictions can configure the platform to provide any subset of the three modes of operation listed below:

**Online blank ballot delivery.** The voter downloads a blank ballot corresponding to their home address and/or party affiliation. The ballot is delivered as a

PDF file. Most jurisdictions instruct voters to print it, mark it manually, and physically return it to the election authorities.

**Online ballot marking.** Voters use the website to mark their ballot selections and download the completed ballot as a PDF file. Online marking makes it easier for voters with certain disabilities to fill out their ballots independently. It also allows the website to prevent overvotes and to warn voters about undervotes, reducing errors. The resulting PDF file can be printed and returned physically. Some jurisdictions, including Delaware, also give voters the option to return it via email or fax.

**Online ballot return.** In some deployments, voters can use OmniBallot to mark their ballots and transmit them to the jurisdiction over the Internet through a service operated by Democracy Live. Like in Washington, D.C.'s attempted Internet voting system [323], jurisdictions print the ballots they receive and then tabulate them with other absentee ballots.

### Deployments

Most instances of OmniBallot appear to be hosted at predictable paths of the form https://sites.omniballot.us/$n$/app, where $n$ is the locality's numeric FIPS code [296]. Statewide deployments use two-digit numbers, and counties and cites use five-digit numbers. We visited all pages with these URL formats in May 2020 and found instances for seven state governments and 98 smaller jurisdictions in 11 states.

Nearly all OmniBallot customers offer online ballot delivery, and we found 70 that offer online ballot marking, but only a few appear to allow online ballot return. We found six jurisdictions that enabled the Internet voting option, notably including the states of Delaware and West Virginia. These jurisdictions and their URLs can be found in Table 3-12.

New Jersey had also announced plans to use Democracy Live for online voting [218, 282] and reportedly did use it for local school board elections in May 2020, but we were not able to locate a deployment for the state.

| Jurisdiction | URL |
|---|---|
| Jackson County, OR | `sites.omniballot.us/41029/app` |
| Umatilla County, OR | `sites.omniballot.us/41059/app` |
| Pierce County, WA | `sites.omniballot.us/53053/app` |
| King Conservation District, WA | `sites.omniballot.us/kcd/app` |
| State of West Virginia | `sites.omniballot.us/54/app` |
| State of Delaware | `ballot.elections.delaware.gov/app` |

Figure 3-12: **Jurisdictions Confirmed to Use OmniBallot's Internet Voting**

### The Voter's Perspective

We now describe how OmniBallot works from a voter's perspective. Screenshots in Figure 3-13 illustrate each step. We use the Delaware deployment as a concrete example, noting some of the differences in other deployments where applicable.

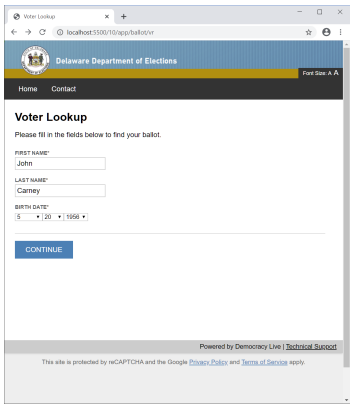
1. **Welcome.** Voters visit the main URL of the website and are greeted by a welcome screen. The voter clicks a button to "Mark My Official Ballot."

2. **Voter lookup.** The voter enters their first and last name and birthdate, and the site locates them in the voter registration database. If multiple voters match, the site lists their street addresses and asks the voter to choose one.

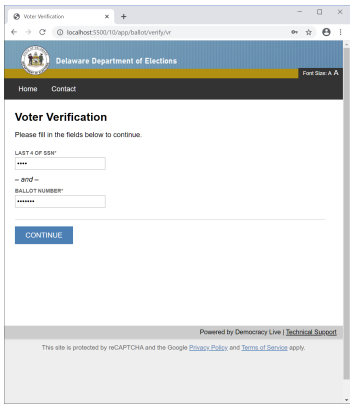
3. **Verify voter.** In Delaware, voters entered the last four digits of their social security numbers and a "ballot number" provided by the state in an email sent by the election administrators. These were verified by the server before the voter is allowed to proceed. Some other deployments we examined did not use this verification step.

4. **Return type.** Delaware let voters opt to return their ballots by mail, by fax, by email (using a webmail portal), or through OmniBallot's Internet voting mechanism ("electronic return"). If mail, fax, or email return was selected, voters could either mark their ballots using the site and generate PDF files to return or retrieve blank ballot PDFs and mark them manually.

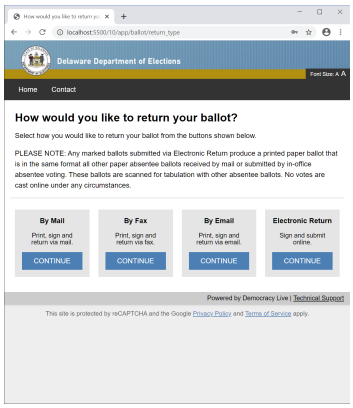
5. **Ballot marking.** The voter can scroll through the ballot and make selections.



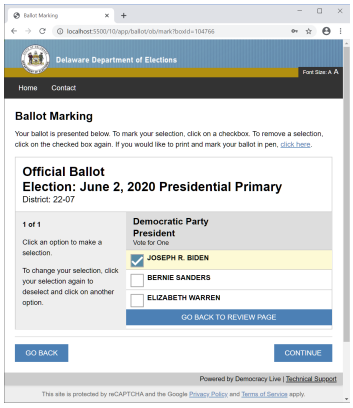

(a) Voter Lookup



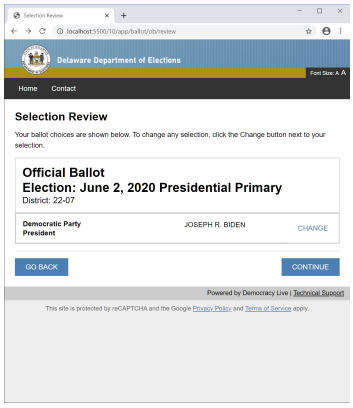

(b) Verify Voter



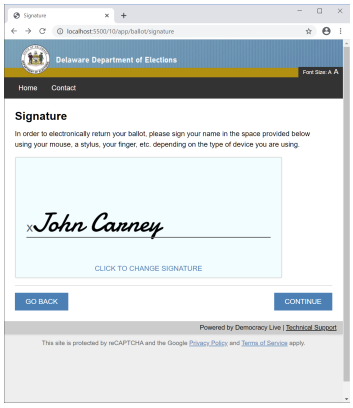

(c) Return Type



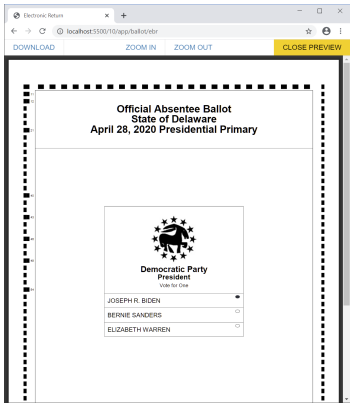

(d) Ballot Marking



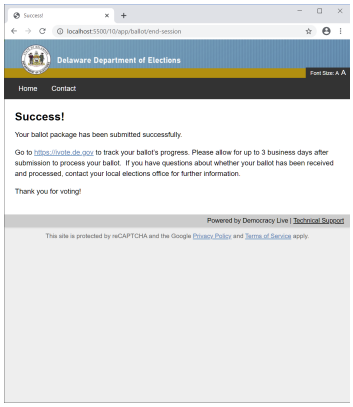

(e) Selection Review




(f) Signature




(g) Preview




(h) Ballot Submitted

Figure 3-13: **Online voting with Democracy Live**, as used in Delaware. The voter's identity and ballot selections are transmitted over the Internet to generate a PDF ballot. Election officials later retrieve the ballot files and tabulate the votes. All screenshots were captured with a local stand-in server.

Write-in candidates can be entered using the keyboard where permitted. The site will refuse to mark more than the allowed number of candidates.

6. **Selection review.** A summary screen shows the selections in each race (or a warning if the voter made fewer than the allowed number of sections). The voter can return to the ballot to change selections or proceed to casting.

7. **Signature.** Voters are instructed to sign their names with the mouse or touch screen, or to type their names. The result is captured as a bitmap image. Some other jurisdictions do not allow a typed signature and instruct voters that their signature must match the signature on file with the jurisdiction.[21]

8. **Electronic return.** Voters are shown a preview of their return packages (which includes their identification information and signature page) and their completed ballot. These are PDF files that the site renders with JavaScript.

9. **Ballot submitted.** When voters are satisfied, they click a button to submit the ballot over the Internet. In Delaware, voters could check whether a ballot in their name has been accepted using their ballot numbers. However, unlike the confirmations provided by E2E-V systems, this mechanism could not protect the ballot selections from modification.

Alternatively, if voters choose to download a blank ballot or to mark a ballot to send via mail, fax, or email, they follow a different path through the site. There is no signature screen after marking the ballot, and instead the voter is provided with a downloadable PDF file of the ballot and return package.

### System Architecture

The web app communicates with several servers to load static files or make API calls, as illustrated in Figure 3-14. Four of these services are controlled by Democracy Live and hosted in Amazon Web Services: `{sites, published, lambda, api}.omniballot.us`; all use Amazon CloudFront as a CDN and have HTTPS certificates for `*.omniballot.us`.

[21] On-screen signatures often differ dramatically from signatures made on paper [110].

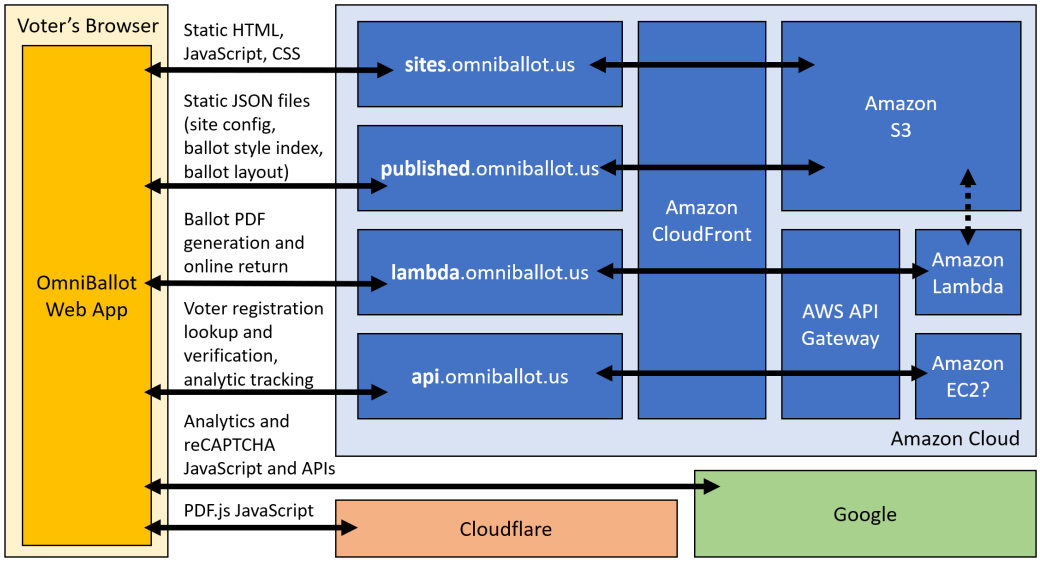



Figure 3-14: **OmniBallot architecture.** The web app runs in the browser and uses HTTPS to load files and call REST-like APIs from several domains. When voting online or marking a ballot, the app sends the voter's identity and ballot selections to Democracy Live services running in Amazon's cloud. The app runs JavaScript loaded from Amazon, Google, and Cloudflare, making all three companies (as well as Democracy Live itself) potential points of compromise for the election system.

The app also loads JavaScript libraries from Google (Google Analytics and reCAPTCHA [12]) and Cloudflare (PDF.js).

The `sites` and `published` servers appear to be backed by Amazon S3. The `sites` server hosts the static HTML, JavaScript, and CSS of the web app, with different paths containing different jurisdictions' deployments or different versions of the code. The `published` server hosts static JSON files that specify the configuration of each deployment (`site-config.json`), provide an index of ballot styles (`lookups.json`), and define each ballot. The `site-config.json` file defines the appearance and workflow of the web app, allowing individual app instances to be heavily customized for each jurisdiction.

The `api` server handles voter lookup and authentication. It provides a REST-like API that allows clients to query for specific voter and ballot information as JSON-encoded HTTP queries and responses. The service is hosted through AWS API Gateway, and may be backed by an Amazon EC2 instance. The `lambda` server uses a similar API format to process ballot PDF generation requests and online ballot return

submissions, and it appears to be backed by code running on the Amazon Lambda serverless computing platform. Calls to both servers include an `x-api-key` HTTP header set to a hard-coded value.

### Client–Server Interactions

In Delaware, the client-server interactions proceeded along the following lines:

1. The browser visits `ballot.elections.delaware.gov` and loads the base HTML page, which defines the site configuration file as `published.omniballot.us/10/site-config.json` and loads the app's base code from `sites.omniballot.us/v9_2_11/combined.js`. The app dynamically loads 24 other JavaScript modules from under the same path. It also loads the Google Analytics library from `googletagmanager.com` and the reCAPTCHA library from `gstatic.com`.

2. The app looks up the voter's registration information by making a POST request to `api.omniballot.us/vr/`*`db`*`/voters/lookup`. This request (and all later POST requests) includes headers for the reCAPTCHA API as an abuse protection mechanism. The request contains the voter's first and last names and date of birth. The server responds with the registration data, including a unique id (*voter_id*), whether the user is a "standard" or military (UOCAVA) voter (*voter_type*), and their party (*voter_party*) and precinct.

3. The app verifies the voter's identity by making a POST request to `api.omniballot.us/vr/`*`db`*`/voter/`*`voter_id`*`/verify`. The request includes the election ID as well as the ballot number and partial social security number entered by the user. If verification succeeds, the server returns a signed JSON Web Token that authenticates the voter_id.

4. To find available elections, the app sends a GET request to `api.omniballot.us/accounts/`*`account_id`*`/currentelections?voter_type=`*`type`*`&voter_party=`*`party`*. The server returns a JSON object for each election with the election name, ID, parent_id, and opening and closing dates. The app then lo-

```
▼ Request Payload    view source
▼ {aid: "10", eid: 1961, packageAid: "10", packageEid: 1961, packageCode: "standard-mail",
    aid: "10"
    eid: 1961
    formFields: {}
  ▶ formTokens: {vid: "vid0", fname: "JOHN", mname: "", lname: "CARNEY", bday: "", House/St
    lang: "en"
  ▶ overlays: {vid: "vid0", fname: "JOHN", mname: "", lname: "CARNEY", bday: "", Precinct:
    packageAid: "10"
    packageCode: "standard-mail"
    packageEid: 1961
    precinctId: 436740
  ▶ selections: [{boxId: 104766, optionId: 161450, selected: true, value: ""}]
    styleId: 293182
```


Figure 3-15: In Delaware, **marked ballot generation** took place on OmniBallot servers. The app sent a POST request (*above*) that included the voter's identity and ballot selections. The server returned the marked ballot as a PDF file. Online voting used a similar request format, with the addition of a browser fingerprint. Marking ballots server-side increases risks to election integrity and ballot secrecy.

cates the appropriate ballot design by loading `published.omniballot.us/10/`*`parent_id`*`/styles/lookups.json`, which is a data structure that associates ballot styles with precincts, parties, and voter types. The ballot itself is defined in a static JSON object retrieved from `published.omniballot.us/10/`*`parent_id`*`/styles/`*`style_id`*`.json`.

5. If the voter chooses to return the ballot via postal mail, fax, or email, the web app generates a ballot PDF file by making a POST request to `lambda.omniballot.us/packagebuilder/v2`. The request includes an HTTP `Authorization: Bearer` header that contains the voter authentication token acquired above. The request body, shown in Figure 3-15, specifies the election, the ballot style, and the voter's name and other registration information. If the voter is marking the ballot, it also includes the ballot selections, encoded as an array of race and selection identifiers. The server returns a URL to a PDF file containing the generated ballot. The file is hosted in Amazon S3, and the URL is a pre-signed object URL [16] with a five-minute expiration.

6. Online ballot return uses a similar API. The app makes a POST request to `lambda.omniballot.us/ebr/build` with the same authorization header. The request contains the same kinds of data as ballot marking, including the voter's identity, registration information, and ballot selections. In addition, the request contains a browser fingerprint generated using FingerprintJS [304] and a base64-encoded PNG image of the voter's signature. The server returns a ballot ID and URLs from which the client can retrieve PDF files of the marked ballot and return package. These are rendered in the browser using the PDF.js library, which is retrieved from `cdnjs.cloudflare.com`.

7. Finally, to submit the ballot online, the client makes a POST request to `lambda.omniballot.us/ebr/submit`, again including the authorization header. The request contains the voter_id and the ballot_id from the previous step, but the ballot selections are not resent. Based on Democracy Live's statements about using Amazon ObjectLock [15], we assume that this API call causes the server to place the return package and ballot PDFs into an ObjectLock-enabled S3 bucket for delivery to election officials. The server sends a response indicating success, and the voting process is complete.

### 3.4.3 OmniBallot Analysis and Attacks

We now assess the security and privacy risks of the OmniBallot platform. We analyze risks created when OmniBallot is used in each of three modes—blank ballot delivery, ballot marking, and online ballot return—and we discuss how (or whether) they can be mitigated. We consider three main classes of adversaries:

**Adversaries with access to the voter's device.** The client-side adversaries with which we are most concerned are ones with the ability to alter the behavior of the voter's web browser, such as by modifying HTTP requests or responses or injecting JavaScript into the context of the site. Several kinds of threat actors have these capabilities, including system administrators, other people with whom the voter shares

the device (e.g., an abusive partner), and remote attackers who control malware on the device, such as bots or malicious browser extensions.

Client-side malware is especially concerning because many devices are already infected by malicious software that could be remotely updated to attack OmniBallot. For instance, Microsoft this year took down a botnet controlled by Russian criminals that had infected more than nine million PCs [267]. Botnets are sometimes rented or sold to other parties to perpetrate attacks [161]. Similarly, researchers recently uncovered more than 500 malicious Chrome extensions in use by millions of people [183], and a popular *legitimate* Chrome extension was hijacked and modified to forward users' credentials to a server in Ukraine [188]. Attackers could use these strategies to target large numbers of OmniBallot voters.

**Adversaries with access to OmniBallot server infrastructure.** The platform's architecture makes server-side adversaries extremely powerful. Depending on which services they compromised, they could change the code delivered to clients, steal sensitive private information, or modify election data, including voted ballots. Potential attackers with such access include: (1) software engineers and system administrators at Democracy Live; (2) insiders at Amazon, which owns and operates the physical servers; and (3) external attackers who manage to breach the servers or Democracy Live's development systems.

**Adversaries with control of third-party code.** Beyond its reliance on Amazon's cloud, OmniBallot incorporates a wide range of third-party software and services, including AngularJS, FingerprintJS, PDF.js, Google Analytics, and reCAPTCHA. Since all this code runs within the app's browser context, it has the ability to access sensitive data or introduce malicious behavior. In recent years, attackers have hijacked several popular JavaScript libraries to target users of software that incorporates them (e.g., [294]). Moreover, OmniBallot clients load some libraries directly from Google and Cloudflare, putting these companies (as well as Amazon) in a position to surreptitiously modify the web app's behavior.

Even large, sophisticated companies are not beyond being compromised by nation states—see, e.g., Operation Aurora, in which China infiltrated Google and a number of other high-tech companies [330]. While Amazon, Google, and Cloudflare have significant incentives to protect their infrastructure and reputations, they also have large stakes in the outcome of major elections, and individual employees or small teams within the companies may feel strong partisan sympathies and have sufficient access to attack OmniBallot. Furthermore, even if these companies' services were perfectly secure against insiders and exploitation, voters may still be distrustful of their ability to handle votes impartially—just as some of the public does not trust the Washington Post under Jeff Bezos's ownership—weakening the perceived legitimacy of elections.

The sections that follow discuss attacks that these threat actors could carry out against OmniBallot's blank ballot delivery, online ballot marking, and electronic ballot return features, and against voters' privacy. We omit some important categories of attacks, including denial-of-service attacks and attacks against voter authentication, due to limits of what we can learn without access to the servers or detailed local election procedures. Table 3.1 summarizes our analysis.

### Risks of Blank Ballot Delivery

OmniBallot's safest mode of operation is online delivery of blank ballots that will be printed, manually marked, and returned physically through postal mail or drop off. (Returning the ballots via email or fax leads to severe risks, which we discuss separately.) Online blank-ballot delivery can provide a valuable enhancement to vote-by-mail systems, but election officials must implement rigorous safeguards to protect against several categories of attacks.

**Ballot design manipulation.** One mode of attack would be to alter the ballot design. For instance, an attacker could change or omit certain races or candidates or substitute a ballot from a different locality. Such changes might be spotted by well informed voters, but other, harder to detect modifications could cause votes to be

counted for the wrong candidate when tabulated by a scanner. For instance, attackers could modify bar codes or timing marks, or shift the positions of selection targets. Conducting these attacks would be straightforward for adversaries with control of the client device, server infrastructure, or third-party code.

To protect against ballot design manipulation, officials first need to check that each returned ballot matches the voter's assigned ballot style, using careful procedures to preserve ballot secrecy. Next, since visual inspection likely cannot detect all modifications that would cause tabulators to miscount the votes, officials either need to count the ballots by hand or manually "remake" the ballots (transfer the votes onto pre-printed ballots) before scanning them. An effective alternative would be to perform a risk-limiting audit [202] (which is necessary in any case to protect against other kinds of error and fraud), but Delaware, West Virginia, and New Jersey do not conduct state-wide RLAs.

| **Configuration** | **Attacker Capability** | | | **Risk** |
|---|---|---|---|---|
| | Manipulate Ballot Design | Compromise Ballot Secrecy | Invisibly Change Votes | |
| Blank Ballot Printing | C S T | | | Moderate |
| Marked Ballot Printing | C S T | C S T | | High |
| Online Ballot Return | C S T | C S T | C S T | Severe |

Table 3.1: **OmniBallot risks.** We show what kinds of attacks are possible when OmniBallot is used in different modes, if an attacker compromises the voter's client (C), Democracy Live's services (S), or third-party infrastructure (T). Ballot designs can be manipulated in all cases. When ballots are marked online, Democracy Live servers see the voter's identity and selections. When ballots are returned online, attackers could potentially change votes without being detected.

**Ballot misdirection.** Another way to attack blank ballot delivery would be to modify the ballot return instructions, rather than the ballot itself, in order to cause voted ballots to be sent to the wrong place or be delayed until too late to count. In Delaware, OmniBallot included the return instructions and a printable envelope in the

same PDF file as the ballot. The attacker could replace the entire delivery address or simply change the zip code or postal bar code to route the ballot to a distant sorting facility. Since OmniBallot verifies the voter's identity before providing the return package, an attacker could decide which ballots to misdirect based on the voter's place of residence or party affiliation.

Voters might detect that their ballots have been misdirected if the jurisdiction provides a ballot tracking service. However, the attacker could simultaneously mail a different ballot in the voter's name—but with votes for the attacker's preferred candidates—reusing the voter's identity information taken from the web app. This would make it appear to voters that their ballots had been received.

Officials can partially defend against misdirection by providing correct ballot return instructions through prominent channels other than OmniBallot, such as on other official sites and in the media. We also recommend that states coordinate with the Postal Service to ensure that postal workers are on the lookout for misdirected ballots.

#### Risks of Online Ballot Marking

Using OmniBallot to mark ballots online, print them, and return them physically raises greater risks than blank ballot delivery. (Again, marking ballots online and returning them via email or fax leads to severe risks, which we discuss separately.) Some of the risks can be mitigated with careful procedures, but others are difficult to avoid, especially if online ballot marking is widely used.

**Enhanced ballot misdirection and manipulation:** OmniBallot's online ballot marking configuration could allow attackers to see the voter's selections before the ballot is generated, allowing them to surgically suppress votes for a particular candidate by misdirecting or modifying only those ballots. The attacker could also reorder the candidates, move the selection targets or timing marks, or encode false votes within barcodes, so that the ballot appears (to a human) to be marked for the voter's selected candidate but will be counted by an optical scanner as a vote for a different candidate. These risks make the procedural defenses discussed in Section 3.4.3 even more crucial

when jurisdictions offer online ballot marking. However, "remaking" the ballot by reading the votes from a barcode, as some jurisdictions do, introduces further security risks, since attackers could change the barcodes without detection. Instead, absent a risk-limiting audit, officials must manually transcribe the human-readable selections to a pre-printed ballot.

**Ballot mismarking:** Online marking enables a simpler style of ballot manipulation that may be impossible to procedurally mitigate: mismark the ballot so that one or more races reflect the attacker's choices instead of the voter's.

Of course, voters could detect this by carefully reviewing their ballots before returning them. However, recent research involving ballot marking devices—which are susceptible to analogous attacks—finds that the vast majority of voters fail to detect errors on machine-marked paper ballots [52]. OmniBallot users who did notice a problem would likely discard the erroneous ballot and use the system to mark another; the attacker could recognize this repeat attempt and mark the new ballot correctly. Even if a few voters alerted election officials, the voters would have no way to prove that the system misbehaved, so officials would have difficulty distinguishing an attack from isolated human error [26].

Prompting voters to carefully review their ballots may increase error detection to a limited extent. However, modeling suggests that the improvement may not be sufficient to detect outcome-changing fraud in close elections unless use of electronic ballot marking is limited to a small subset of voters [52].

**Compromising ballot secrecy:** Online ballot marking carries an elevated risk that attackers could compromise the voter's secret ballot. Attackers with the ability to alter or inject code into the web app could exfiltrate the voter's identity and ballot choices. Moreover, since the web app sends the voter's identity and ballot choices to `lambda.omniballot.us` in order to generate the marked ballot PDF file, an attacker with only passive access to the data processed by this service can learn voters' ballot selections, even when the ballot is returned physically.

Furthermore, the ballot return package, including the voter's identity and marked ballot, is saved locally to the voter's computer before being printed. This creates a risk that client-side attackers, including other local users, could gain access to the file. Even if voters delete the files, forensic tools may allow adversaries to recover the ballots long into the future [137].

### Risks of Online Ballot Return

OmniBallot's online ballot return mode carries similar risks to online ballot marking as well as severe additional risk that cast votes could be changed at large scale without detection. These risks cannot be adequately mitigated with procedural changes or readily available technology.

**Lack of end-to-end verifiability:** OmniBallot does not attempt to achieve E2E verifiability. Instead, it uses a protocol that provides no way for voters, officials, or Democracy Live itself to verify that the ballot selections a voter chooses are the same as what officials receive. Consequently, an attacker with control of the voter's client, of Democracy Live's infrastructure, or of any of the third-party services from which the client loads JavaScript, could change recorded votes. Unlike ballot marking with physical return, where the voter has a chance to review the printed ballot that is sent for tabulation, voters have no practical ability to detect vote-changing attacks involving online ballot return. Nor do election officials. Democracy Live itself would have little opportunity to detect attacks that were perpetrated by client-side malware or third-party infrastructure.

**Vote-changing attacks:** Recall that OmniBallot's online voting is accomplished by making two API calls to `lambda.omniballot.us`: one that submits the voter's identity and selections and receives a ballot ID and a URL for the marked ballot PDF file, and another that submits the ballot ID and causes the ballot to be delivered to election officials. Both requests are authenticated with a bearer token that is provided after checking the voter's identity.

One way to subvert this process would be to inject malicious code into the web app. This could be accomplished with local malware (such as a malicious browser extension) or by delivering malicious code as part of the JavaScript that OmniBallot loads from Amazon, Google, and Cloudflare servers. Insiders at these companies or at Democracy Live could attempt such an attack, as could external attackers who compromised any of the companies' infrastructure.

Once in control of the client, the attacker could cause the web app to substitute ballot selections of the attacker's choosing. To hide the changes from the voter, the attacker would simply have to generate a separate ballot PDF file to display to the voter that *did* match the voter's selections. This could be accomplished by modifying the real ballot PDF file using client-side code. As a result, the web app would show a ballot containing the selections the voter intended, but the ballot that got cast would have selections chosen by the attacker. The attack would execute on the client, with no unusual interactions with Democracy Live, so there would be no reliable way for the company (or election officials) to discover it.

Attackers with control of the `lambda.omniballot.us` service—such as malicious insiders at Democracy Live or at Amazon, or external attackers who penetrated either company's systems—would have a separate way of changing votes. Malicious code on this server could return one PDF to the voter and store a different one for delivery and counting. Voters would have no way to notice the change.

### Risks of Email-Based Ballot Return

Like other modes of online voting, email-based ballot return faces severe security risks that cannot be adequately mitigated with available technology or controls. Different OmniBallot jurisdictions use widely varying procedures for email-based return; here we focus on the way it is implemented in Delaware. Even after discontinuing OmniBallot, Delaware allowed voters to return ballots by email.

Delaware voters who choose to return their ballots by email are instructed to use Egress Switch [283], a "secure email" platform produced by U.K.-based Egress Software Technologies, Ltd. Rather than directly emailing the ballot, voters visit

| Voter Private Information | Configuration | | |
|---|---|---|---|
| | Blank Ballot Delivery | Online Ballot Marking | Online Ballot Return |
| IP address/coarse physical location | ✓+ | ✓+ | ✓+ |
| Delaware voter ID number | ✓+ | ✓+ | ✓+ |
| Name, address, and date of birth | ✓* | ✓* | ✓* |
| Party affiliation | ✓* | ✓* | ✓* |
| Partial social security number | ✓ | ✓ | ✓ |
| Vote selections | | ✓ | ✓ |
| Browser fingerprint | | | ✓ |

Table 3.2: **Access to privacy-sensitive data.** We show what data is shared with Democracy Live when using OmniBallot in each mode offered in Delaware. A + indicates that the information is also sent to Google; a * indicates that Google can infer it. All data is implicitly sent to AWS.

`switch.egress.com` and sign up for accounts using their email addresses. After proving that they have received a confirmation code sent to that address, the voter can log in to a webmail interface, compose a message to a Delaware elections email address, and attach the voted ballot as a PDF file. The recipient receives an email notification that the message is available and can log in to the same system to retrieve it.

A full analysis of Egress Switch is beyond the scope of this analysis, but we note that it is effectively serving as a second Internet voting platform, with broadly similar risks to OmniBallot's online return mode, including a reliance on large tech companies for trusted infrastructure. Egress appears to be hosted in Microsoft's cloud and to store encrypted messages in Amazon S3 servers located in the U.K. Routing domestic voters' ballots through a foreign jurisdiction may weaken the legal protections surrounding ballot secrecy and exposes voters to a greater risk of surveillance or other attacks by a foreign government [94].

Depending on the voter's existing email provider, Egress Switch may offer privacy advantages, particularly as the sender may only view sent messages for a limited time.

On the other hand, it centralizes voted ballots on a single third-party platform, which must be trusted to deliver them without modification. As with OmniBallot, Switch itself, and the third-parties it trusts, can see and change the ballot before it is delivered, and there is no apparent mechanism by which voters can independently confirm that their voted ballots have been received by election officials without modification.

### Risks to Voters' Privacy

OmniBallot has access to a large amount of privacy-sensitive data (see Table 3.2): voters' names, addresses, dates of birth, party affiliations, and other voter registration fields; their coarse physical locations from their IP addresses; their partial social security numbers; and, in either the ballot marking or online voting configurations, their actual ballot selections.

In addition, when votes are cast online, OmniBallot's client-side code takes a fingerprint of the browser and sends it to the server with the voter's registration data and ballot selections. If Democracy Live shared this data with other sites, they could recognize the voter's browser and associate it with their identity and votes. Browser fingerprints are incredibly privacy invasive [115]—they can uniquely track a browser even after the user has taken defensive measures such as clearing cookies, as well as between private browsing and normal browser modes [327].

This data about the voter would be valuable to many parties: advertisers, political candidates, or attackers seeking to conduct disinformation campaigns. Notably, Democracy Live appears to be silent about whether, or for how long, they store this data, how they use it, or whether it will be shared or sold to third parties. Prior to our work, OmniBallot included no terms of service or privacy policy (though it did link to Google's, as sites that use reCAPTCHA are required to do).

OmniBallot also makes extensive use of first- and third-party tracking mechanisms to monitor voters' interactions with the platform. It sends Google Analytics extensive browser configuration information, the URLs of pages the voter visits within the app, whether they are a UOCAVA voter, and the voter's ID number. In Delaware, the same ID number is used in the state's publicly available voter file, where it is associated

with the voter's full name, address, phone number, birth year, and party. Google could use the ID field to personally identify the voter and potentially to associate the voter's identify with other tracking cookies.[22]

### Risk Summary

Below, we briefly summarize our findings concerning OmniBallot's three main modes of operation. Our assessment of their relative risk accords with recent guidance by the U.S. Cybersecurity and Infrastructure Security Agency [297, 311].

**Blank ballot delivery.** When OmniBallot is used to deliver blank ballots for printing, attackers could modify certain voters' ballots or return instructions to omit candidates, cause votes to be scanned incorrectly, or delay or misdirect mail-in returns. These risks can be largely mitigated with rigorous election procedures, and, with such protections in place, we consider the overall risk to be moderate.

**Online ballot marking.** Using OmniBallot to mark and print ballots carries greater risks. Attackers can learn the voter's selections and target ballots for a disfavored candidate by misdirecting them or causing them to be scanned as a vote for somebody else. Attackers could also mark the ballot for different candidates than the voter intended, which, although visible, many voters would likely fail to detect. Voter education and procedural defenses can only mitigate these attacks to an extent, so we consider the risk to be high. As the risk further increases when online marking is widely used, we recommend limiting its deployment.

**Online ballot return.** When ballots are returned over the Internet using OmniBallot, there is no way for voters to confirm that their votes have been transmitted without modification, and attackers could change votes in ways that would be difficult for voters, officials, or Democracy Live to detect. Attacks could be conducted through client-side malware, compromise of third-party services such as Amazon and Google,

[22] This behavior appears to be in violation of the Google Analytics terms of service [147], which prohibit sending personally identifiable information to Google.

or infiltration of Democracy Live. Administrative controls and audits cannot prevent such attacks. Given the possibility for undetected changes to election results, we consider the risks of online voting to be severe.

## 3.5 Discussion & PostScript

Here we provide an in-depth discussion of the lessons learned from examination of both systems. We begin by discussing how the failures of transparency and auditing significantly hindered our analysis, demonstrating the need for transparency and accountability in labeling as discussed in Chapter 2. We then continue with a discussion of various technical recommendations for vendors in this space, as well as elections administrators and policymakers. We hope that this analysis helps inform better policy, and ameliorates the misaligned incentives in this market.

### 3.5.1 Transparency and Auditing

Put in the frame of Chapter 2, the actions of the developers in both cases exacerbated the information asymmetries between all actors in the market. Neither the elections officials (who purchased said products) or the software vendors could be held to account. As a result, both companies were allowed to over promise and undertreat, providing minimal security and harming user privacy. We further explore this argument below.

**Claims of Security are Largely Poor Labeling:** Neither company provided adequate documentation of their system design, and what little public information was provided had been littered with confusing or meaningless claims of security. Voatz's FAQ, for example, stated that their system was "doubly anonymized," used "end-to-end vote encryption" and "anonymized voter-verified digital receipts," and later leveraged a "voter-verified audit trail" — none of which have any definition in the cryptographic literature, and it is still unclear what these statements are referring to even after our full analysis of their system. Perhaps more bizarrely, Democracy Live argued that their online ballot return capability *should not be considered Internet voting at all*,

perhaps attempting to avoid the scrutiny required for such systems. Whatever the motive, this kind of obfuscation makes it difficult for voters, election officials, and other policymakers to understand whether the technologies are safe.

**Disclosure Policies Limit Transparency and Accountability:** Coordinated vulnerability disclosure and bug bounties can be incredibly powerful tools for transparency and accountability, allowing security researchers to provide feedback to the company and safely disclose their findings while forcing the company to fix the discovered bugs. Taken in the economic context (as in Chapter 2), such mechanisms can help regulate markets for credence goods, and voluntary use can be a powerful argument in favor of effective self-regulation. Unfortunately, Democracy Live and Voatz serve as worked examples of how such policies may be more signaling than function.

As previously mentioned, we analyzed the most recent version of Voatz's app available in the Google Play store as of January 1, 2020. At the time, Voatz also provided a "bug bounty" version of the app via a third party service called HackerOne [307]. The company touted the bug bounty as evidence of their commitment to independent audits, as well as "community vetting" of the product [308]. We chose not to examine this version of the app for several reasons.

First, evaluating the bounty app alone would have introduced additional threats to validity, and as the differences between this version and the ones that have been fielded are unclear, we chose to err on the side of realism. Worse, all apps are independently randomly obfuscated such that static analysis of each requires a lengthy manual deobfuscation process, so repeating this work on a second app represented significant additional effort.

Second, crucially, the bounty did not provide any additional helpful insight into Voatz's server infrastructure, nor did it provide any source or binary for the API server to test against. Indeed, when the decision to analyze the live app was made, both Voatz's bug bounty app and the Google Play app failed to connect.

Finally, the terms of the bug bounty contained untenable restrictions that hinder

an open dialog about the system. For example, the bug bounty excluded both MITM attacks and attacks requiring physical access to the device. This physical access restriction could be read to exclude all of our on-device attacks — To simulate an attacker with access to a remote root-level vulnerability, we used a manual jail-breaking technique which happens to require physical access. The MITM restriction would similarly put the sidechannel attack, as well as the analysis of an adversary that controls Voatz's API server, explicitly out of scope. Worse, the bug bounty, in coordination with their "responsible disclosure policy," also denies researchers safe harbor unless they wait to disclose their findings until some arbitrary time that Voatz decrees the bug fix to be fully deployed [310].

In short, Voatz's bug bounty appeared to restrict the researcher from disclosure, failed to provide adequate resources for analysis, and arbitrarily considered whole classes of realistic vulnerabilities outside of the scope of the exercise. We concluded that the bug bounty is not particularly relevant for allowing researchers to vet, audit, or improve the system's security, and serves as an example of how such engagements may not be as effective as one may hope. Democracy Live's approach was far more straightforward, albeit somewhat worse than Voatz's. They provided no system to test, and the company's reporting guidelines at the time of our analysis (Fig. 3-16) prohibited further disclosure of reported problems without their permission. After we made our findings public, they adopted a new policy [101] modeled after Disclose.io's Coordinated Vulnerability Disclosure Template [108]. However, the new policy permits disclosure post-mitigation, but there are no set timelines nor any apparent recourse if the company excessively delays or chooses not to fix a problem.

If the goal is to maximize the utility of audits and increase transparency through a bug bounty or public disclosure policy, vendors should provide source code for both the server and client, publish full system implementation and operational details, and explicitly free researchers to divulge their findings after the industry-standard 90 days, or, at the very least, on a fixed, publicly-available time schedule. As it stands, these policies may discourage responsible disclosure and could prevent researchers from alerting officials or the public about flaws that go unfixed.

## Reporting Bugs

Democracy Live considers keeping the voting process secure and confidential as our highest priority. We work with industry leading experts in information security in both public and private sector to maintain the privacy and integrity of our customers data as well as that of their voters.

We encourage security professionals and developers of all skill levels to help us in keeping our OmniBallot platform secure for all voters. If you believe you've discovered a bug or vulnerability while receiving your ballot through OmniBallot, we ask you contact us immediately.

### Submitting Bug or Vulnerability Reports

Bug reports can be sent to security@democracylive.com with the subject line of Bug Report – DD-MM-YYYY

By submitting a report, you agree to not disclose that bug to any third parties without the approval of Democracy Live. We ask that bug reports follow the standard Github format:

Figure 3-16: **Democracy Live's vulnerability reporting guidelines** stipulated that researchers who reported problems could not further disclose them without permission. Although it is unclear if this policy is enforceable, such restrictions run counter to best practices and may chill responsible disclosure.

**Failures of Transparency Hinder Independent Analysis & Degrade User Privacy:** The lack of public source and incomplete documentation exacerbate many of the security and information privacy risks presented in this chapter, and serve as an example of the importance of transparency in election software in particular. While we had to expend considerable time and effort to deobfuscate each system's client-side code and make the results accessible for analysis, the flaws themselves are hardly novel — sidechannel attacks, for example, are well known in the cryptographic engineering and research literature, and many of the other issues appear to be the result of poor design and nonstandard implementation. Open access to code, each systems' design, and running test implementations would have likely revealed these flaws rapidly and encouraged these companies to fix them, or at least dissuade election officials from putting the voting public at risk.

It is also clear that this lack of transparency did not significantly hinder our ability to discover the flaws presented in this chapter, and will similarly fail to prevent a well-resourced adversary from doing the same. In our analysis, we never intentionally connected to either system's servers, and were forced to retarget all communication to our own infrastructure both to avoid disrupting their systems and to comply with the

law. Criminals or foreign intelligence agencies, on the other hand, are not constrained to follow U.S. law and would likely have no qualms about disrupting normal operations, including by connecting to elections servers or attacking the company directly. Such adversaries will therefore have an *easier* time discovering exploitable vulnerabilities, and are more free to explore flaws we were unable to investigate; it is possible that these systems' server infrastructure, third party infrastructure, elections administrator interfaces, and other parts of their service have issues that are impossible to analyze without further access.

Finally, the lack of explicit disclosure specifying exactly what voter information is collected, how it is used, how long it will be retained, and what third parties may have access constitutes a sharp deviation from privacy best practices, and is an especially concerning omission given the sensitivity of voting information.

For example, despite having access to a wide range of sensitive personally identifiable information, OmniBallot had no privacy policy, leaving voters uniformed about what legal limitations, if any, restrict the company's use of this data. For example, it remains unclear whether the company could legally share such data with political campaigns, law enforcement, foreign governments, or ad tech companies. Moreover, due to OmniBallot's reliance on third-party services, Amazon and Google store or receive some or all of this data. Statutory requirements, Democracy Live's contracts with third parties, and contractual obligations to election jurisdictions may offer some legal protections, but these are largely invisible to voters.

While Voatz *did* have a privacy policy, its lack of transparency on important privacy practices such as third-party data sharing left voter data unprotected. Beyond serving as a notice to consumers, privacy policies are a critical part of the privacy protection framework, especially in jurisdictions such as the United States that lack comprehensive privacy laws; individual commercial privacy is generally protected in the U.S. only if companies make concrete commitments in their stated privacy policies [276]. For example, because Voatz did not place any explicit data retention time limits on their third-party ID verification service in a publicly-visible privacy policy, users are at risk of having such sensitive election-related information held indefinitely. Barring

local statutory restrictions and/or contractual obligations unknown to the authors, the lack of a concrete privacy policy renders Voatz and their partners unaccountable for such privacy failures, and makes it unclear if Voatz can use the information outside of the context of the election itself.

As mentioned in Section 3.3.2, the only notification to the user that their ID was being sent to a third party in Voatz's app is the faint logo placed in the lower right corner of the app's photo screen, and we found no user-accessible indication of the use of various other third parties at all. While the privacy policy does state that Voatz "may transfer Personal Information to third parties for the purpose of providing the Services," it never discloses what information or to whom. Without knowledge of where their personal information is going, there can be no informed consent — as it stands, even the most diligent and privacy-focused individual is likely to misunderstand and assume that their data, particularly their ID information, is only being shared with Voatz.

### 3.5.2 Responsible Disclosure Process and Vendor Reaction

Not only did these disclosure policies limit our analysis, requiring extensive reverse engineering and reimplementation, they also further complicated our ability to report the issues we discovered.

Given the heightened sensitivity surrounding election security issues, and due to concerns of potential retaliation, in both cases we chose to alert the U.S. Department of Homeland Security (DHS) and anonymously coordinate disclosure through their Cybersecurity and Infrastructure Security Agency (CISA). In the case of Voatz, before publicly announcing our findings, we received confirmation from the vendor through CISA, and, while they disputed the severity of the issues, they appeared to confirm the existence of the side channel vulnerability, and the PIN entropy issues. In the case of Democracy Live, we discussed our findings directly with the company's management team. With both studies, we also spoke directly with affected election officials in an effort to reduce the potential for harming any election processes.[23]

[23]In both of these cases, the vendor shared additional information, but, as those details were part

### Democracy Live's Reaction

At our recommendation, Democracy Live made some limited improvements, though there is a far amount of work to be done. For example, Democracy Live has since posted a privacy policy that covers all OmniBallot instances and prohibits the company from using voters' information for any purpose unrelated to servicing their ballots [105]. However, the policy does not provide explicit limits and guarantees about the retention, protection, and disposal of this data.

The company's reporting guidelines at the time of our analysis (Fig. 3-16) prohibited further disclosure of reported problems without their permission. After we made our findings public, they adopted a new policy [101] modeled after Disclose.io's CVD template [108]. The new policy permits disclosure post-mitigation, but there are no set timelines nor any apparent recourse if the company excessively delays or chooses not to fix a problem. These policies may still discourage responsible disclosure and could prevent researchers from alerting officials or the public about flaws that go unfixed.

To the best of our knowledge, Democracy Live has never denied or contested the findings of our analysis. However, in a podcast on election integrity, Democracy Live's CEO Bryan Finney made the following statement:

> "I think it can be extremely valuable, and a healthy tension between [us and] academics, you know, it's their job to break stuff. They want to go make sure it's a secure system ... they want to push the innovators and the providers. It's our job to go build it, it's their job to go see if they can break it, and that's a healthy balance.
>
> Where it becomes unhealthy, in my perspective, especially from an academic standpoint, is when you go in subjectively, not objectively ... And, although there is no secure, 100% technology — you put your ballot in the mailbox how secure is that? — we happen to use a federally approved cloud environment, that in theory is more secure than a fax machine ... The

of confidential communications in the vulnerability disclosure process, they are not included in this chapter. Nothing provided by either vendor contradicted any our factual findings in either study.

> challenge is when you have subjective either academics or antagonists or advocates who don't want to engage, they don't want to collaborate. You know, we had an experience where some folks instead of reaching out to us to talk to us about system and how it works they went right to a national media outlet. That, to me, doesn't strike me as objective research, there's a subjective opinion going into that effort. [65]"

Delaware and New Jersey discontinued use of OmniBallot for online voting [271], but Delaware continued to allow webmail-based ballot return, and West Virginia continued to use the system as-is in both the primary and 2020 presidential elections.

### Voatz's Reaction

A preprint reporting the Voatz-relevant results of this work was publicly disseminated on February 13th, 2020 and covered in press reports [264]. As a result of our findings, Mason County, Washington, announced it would discontinue using Voatz, followed quickly by West Virginia [87].

Instead of addressing the reported vulnerabilities, Voatz responded by attacking the credibility of this analysis. In both a public press call [309] and in a blog post entitled "Voatz Response to Researchers' Flawed Report," [306] company officials downplayed the severity of the findings, impugned our intent as well as our methodology, and claimed (incorrectly) that we had examined an outdated version of the app — but oddly never denied the findings themselves.

On March 13, 2020, Trail of Bits, a third-party security firm, released a document detailing a white-box security analysis of Voatz [54]. Their work cited our original analysis, confirmed the veracity and severity of all findings reported here, and explicitly contradicts Voatz's criticism — supporting our methodology as an industry-standard process and affirming that there were no security relevant differences between the app we examined and the internal master. Trail of Bits also confirmed that the server-side code contained further vulnerabilities opaque to us (finding 48 issues in total) and that Voatz's protocol is not E2E-V, found no evidence of the mixnet claimed by

Voatz, and reported that Zimperium was entirely disabled in at least one of their most recent elections. Remarkably, this report was *not* the result of an independent audit — the firm's work appears to have been funded by Voatz. Finally, HackerOne has since removed Voatz's bug bounty from their platform — a company first — citing concerns around Voatz's apparent inability to interact in good-faith with security researchers [205].

Despite the findings of their own self-funded audit explicitly confirming the results in our work, Voatz's CEO continues to publicly deny the veracity of our findings, claiming that "there are like so many errors in the MIT report, that it's just really really hard to accept that report" [130].

### 3.5.3 Technical Recommendations

Based on our analysis of these two systems, we offer a series of recommendations for election administrators, policymakers, and vendors in order to help protect the integrity of elections conducted using remote accessible voting systems and safeguard voters' privacy. These are in addition to the procedural defenses discussed in Section 3.4.3. Many of these recommendations apply more generally to all systems for online voting or ballot delivery and marking that jurisdictions may be using or considering.

**Eliminate electronic ballot return.** Online ballot return runs counter to the clear scientific consensus, as expressed by the National Academies [224], that the Internet should not be used for the return of marked ballots. Our analysis shows that votes cast online using either system could be surreptitiously changed without voters, officials, or the vendor being able to detect the attack. Given the risks, we recommend that elections administrators refrain from using online ballot return, including ballot return via email. Instead, administrators should focus on improving the efficiency and accessibility of physical ballot return paths, which carry fewer risks of large-scale manipulation.

**Limit the use of online ballot marking.** In the ideal case, online ballot marking provides valuable usability and accessibility benefits. For absentee voters with disabilities that make it impossible to mark ballots by hand, such a tool could provide greater independence and privacy. At the same time, it carries higher risks of ballot misdirection, manipulation, and mismarking than blank ballot delivery, and research with ballot-marking devices suggests that most voters will fail to spot altered ballots, even if prompted to check [52]. As online marking becomes used more widely, it becomes a more attractive target, and the risk that attacks could change election outcomes increases rapidly. For these reasons, we recommend offering online marking only to voters who could not otherwise mark a ballot independently, and not to the general public. Furthermore, marked ballots should always be printed and physically returned.

**Mark ballots using client-side code.** OmniBallot's design, as used in Delaware, creates unnecessary risks to ballot secrecy and integrity by sending the voters' selections, coupled with their identities, to an online service when generating marked ballots. These risks could be avoided by marking ballots locally in the browser, using client-side code.

Democracy Live already offers an option to do this. OmniBallot deployments in California, Virginia counties, and Washington, D.C. use an alternative online marking approach called "Secure Select," in which marked ballots are generated without sending selections to a server [275]. After downloading the return package, the voter is redirected to a page on `ss.liveballot.com`, which delivers JavaScript for generating the marked ballot entirely within the browser.

In addition to Delaware, jurisdictions in Colorado, Florida, Ohio, Oregon, Washington State, and West Virginia appear to use the more dangerous server-side marking mechanism. We recommend that they switch to client-side marking.

**Implement risk-limiting audits.** When a digital system is used to deliver blank ballots that are marked by hand and physically returned, this generates a strongly

voter-verified record of voters' choices. However, attackers can still manipulate the ballot design in ways that would cause votes to be miscounted when tabulated by an optical scanner. To mitigate this, we recommend that officials perform risk-limiting audits (RLAs) [202], which limit the probability that the election outcome differs from the outcome that would be found by a full hand-count. As with in-person voting, RLAs are an essential defense against error and fraud.

**Reduce unnecessary trust in third parties.** Both systems' security depend not only on the security of the vendor's code and procedures, but also on the security of services provided by a number of third parties, including Amazon, Google, and (in the case of OmniBallot) Cloudflare. Attackers that breach their systems (or rogue employees within the companies) could alter votes that are returned electronically. Limiting trust can reduce this risk, to an extent, by removing inessential dependencies (e.g., Google Analytics) and applying subresource integrity [14] to static libraries (e.g., PDF.js). However, eliminating all reliance on third-parties may be inadvisable, as it is difficult, if not impossible, for small companies like Voatz or Democracy Live to deliver the same level of infrastructure security and resilience as a leading cloud provider.

**Increase transparency and facilitate independent review.** Unlike in-person voting equipment, which is tested by federally accredited labs for compliance with the EAC's Voluntary Voting System Guidelines [299], there are no federal standards or certification processes for platforms like Voatz and OmniBallot. This means local and state officials are largely dependent on the vendors themselves when assessing such products. Officials should insist that all digital voting systems be subjected to public examination by independent security experts before considering them for use. Such evaluation has exposed critical vulnerabilities in Internet voting systems in the past (e.g., [323, 157]), preventing flawed technologies from putting elections at risk. That either system has been used before without reported problems — predominately for small populations and for low-risk blank-ballot delivery — does not establish that it

can be used safely for online voting or with large numbers of voters in high-stakes elections.

To facilitate independent analysis, we recommend that all parties adopt a vulnerability disclosure policy that follows best practices, such as NTIA's CVD policy template [232], and make all source code available for scrutiny.

## 3.6 Conclusion

Beginning with West Virginia, Utah, and Colorado, the U.S. has ventured down the path of Internet voting. Despite the concern expressed by experts, companies are selling the promise that these systems are "secure" enough for practical use.

Yet our analysis has shown that these systems are *not* secure, and suffer from extensive privacy failings. In one system, even a passive network adversary can discover a user's vote, and an active one can disrupt transmission in response. For either system, an attacker that controls a user's device also controls their vote, easily brushing aside any built-in countermeasures. And our analysis shows that any actor that controls these systems' servers — including many third parties — likely have full power to observe, alter, and add votes as they please.

A natural question may be why such services have been fielded in the first place. Speaking to the Harvard Business Review, Bradley Tusk, a political philanthropist whose foundation was responsible for funding a number of these systems' deployments, stated:

> "It's not that the cybersecurity people are bad people per se. I think it's that they are solving for one situation, and I am solving for another. They want zero technology risk in any way, shape, or form. [...] But in my view, then you can't resolve the issues on guns, on climate, on immigration, because the middle 70% doesn't participate in primaries [...] I am solving for the problem of turnout [314]."

While we appreciate and share Tusk's desire to increase voter participation, we do not agree that the security risks in this domain are negligible; we believe that

the issues presented in this work outweigh the potential gains in turnout.[24] As we have shown in this chapter, vulnerabilities inherent in these systems and the problems caused by a lack of transparency and accountability for basic failures are very real. The choice here is not about turnout, but about an adversary controlling the election result and a loss of voter privacy, impugning the integrity of the election as a whole.

Given the severity of failings discussed in this chapter, the lack of transparency, the risks to voter privacy, and the trivial nature of the attacks, we suggest that any near-future plans to use either system for Internet voting in high-stakes elections be abandoned. We further recommend that any future designs for voting systems (and related systems such as e-pollbooks) be made public, and that their details, source, threat model, as well as social and human processes be available for public scrutiny.

Note that all attacks presented are viable regardless of the purported use of advanced management tools and cryptography. We join other researchers in remaining skeptical of the security provided by blockchain-based solutions to voting [99, 251, 236], and of internet voting in general [224], and believe that this serves as an object lesson in security — that the marketed use of a series of tools does not indicate that a finished solution provides any real guarantees of security.

Finally, similarities between these two analyses gives credence to the argument that current market incentives do not favor security for such systems. OmniBallot and Voatz represented a near totality of the U.S. market for internet voting systems, had similar failures of design and disclosure, continue to make unclear or incorrect claims about their systems, and have made vague statements disputing the relevance of our findings despite the overwhelming proof presented in this chapter. Worse, these are not isolated examples: Past security analyses of Internet voting systems deployed abroad, including those used in Moscow [138], Australia [157], and Estonia [280], similarly found significant risks to election integrity and faced complications due to policy-based hurdles to analysis, ranging from incomplete documentation and lack of source code to restrictive vulnerability disclosure policies.

[24] Indeed, it is unclear if mobile and internet voting actually increases voter turnout. A study from Switzerland [143] finds, somewhat surprisingly, no statistically significant increase in voter participation.

These works provide concrete evidence to policymakers that regulatory intervention is necessary to avoid allowing this market failure to continue negatively impacting the integrity of our democratic process. We must incentivize vendors to remove barriers to public scrutiny, provide stringent security standards and labels, and create legal regimes for holding these vendors liable for bad behavior. In short, we must shift the burden of proof to the developer to demonstrate that their system is secure, to both the public and the security community, *before* it can be trusted as a crucial component in the democratic process.

# Chapter 4

# Deniable Messaging

In the last chapter, we demonstrated that the structure of a market can degrade the security properties of deployed systems. Here, we consider whether the inverse is also true — that the design of systems could alter attackers' incentives and thus provide better security. It is clear that cryptographic protocols, in particular, can incentivize or disincentivize bad behavior.

Take, for example, the cryptography used for verifying email. Email has long been the world's largest messaging scheme, used ubiquitously for personal, industry, and government communication. As such, it is a valuable target for attack: a user's account is a trove of sensitive information, unauthorized access to which enables spam, fraud, blackmail, and other abuse.

To help protect users from spam and fraud, the IETF developed a widely-adopted standard called DomainKeys Identified Mail (DKIM) [72]. DKIM's goal is to assure the receiving server that each incoming message was really sent from the domain it appears to be from, enabling inter-domain accountability in case of spam and easy detection of spoofed messages. DKIM's protocol is simple: the originating server cryptographically signs each outgoing email's contents and metadata, allowing the receiving server to verify the message after looking up the sending server's public key via DNS.

While DKIM was an important innovation that continues to be critical to the email ecosystem, its design came with an unintended side-effect: namely, email thieves can

credibly convince any third party that stolen messages are authentic and unmodified via DKIM signatures from a reputable service provider. **This effectively increases the incentive for attackers to target and break into email accounts, as a successful attacker can credibly (and anonymously) sell, publish, or use the stolen data for blackmail.**

Email attributability has had real-world impact. For example, Wikileaks publicly asserts [321] that it relies on DKIM signatures to confirm the veracity of their publications: Wikileaks leveraged DKIM to authenticate messages stolen from the Democratic National Committee (DNC) and Hillary Clinton's campaign chairman during the 2016 U.S. presidential election season [318]. Because of DKIM, any third party could easily confirm the legitimacy of these stolen messages using public keys tied to Google and Microsoft's email services, despite the information's questionable origin. Indeed, the practice of using DKIM to verify unauthorized email leaks has now become a standard journalistic practice [268, 213], with the Associated Press releasing a software tool for this purpose [33].

DKIM's attributability problem has been recognized but unsolved for some time. Jon Callas, one of the original authors of the DKIM RFC, has publicly stated that attributability is an unintended design flaw of the protocol [74, 75], and has since suggested a number of ways to mitigate its impact, but notes that proposals at the time of his writing were insufficient or impractical [73]. Other researchers also flagged the issue as early as 2004, e.g., Adida *et al.* [9], Unger *et al.* [302], and Bellovin [44]; however, designing a practical, non-attributable DKIM replacement has remained an open question.

It is alarming that an unintended result of an ubiquitous messaging protocol has produced a *scalable, by-default* system for credible propagation of illicitly obtained private messages. The specific DNC incident might well have happened with or without DKIM: for a high-value target, interested parties would likely seek to verify the stolen emails in various ways, including non-technical methods (e.g., journalistic corroboration, cross-checking timestamps, geolocation, etc). But just the possibility of manual verification — a possibility that has existed since handwritten letters — is

a stark contrast from the easy, inbuilt attribution that has *unintentionally* become ingrained in today's email ecosystem.

Public figures are not the only victims of email breaches; new reports of email theft seem to surface every few weeks. Astoundingly, all of Yahoo!'s 3 billion email accounts were compromised in a 2013 breach [285]. Although Yahoo!'s users have been spared public dissemination of their messages, others (e.g., Sony and Stratfor), have been less fortunate [320, 319]. Attackers appear to have diverse motives, ranging from financial gain — e.g., selling patient healthcare data gleaned from emails [164] — to industrial espionage and monitoring political dissidents and foreign officials [287].

In light of the potential harm to users, it would be irresponsible to let DKIM's unintended side-effect of attributability remain unscrutinized: if attributability is to remain a feature of DKIM, it should be as a result of a deliberate decision that takes into account the range of technically feasible alternatives. With the above as motivation, we ask:

> *Is it possible to mitigate the potential harms of attributability in DKIM while maintaining the system's efficient spam and spoofing resistance?*

An initial intuition may be that attributability of stolen email is an unavoidable side effect of spam and spoofing resistance, given the indirect and decentralized nature of email: it is intuitively unclear how a recipient with no communication to the sending server can be certain of a message's origin without also gaining the ability to convince a third party of the same. Under certain conditions, this intuition amounts to an impossibility. Yet, perhaps surprisingly, our work shows that modern cryptography can reconcile the apparently conflicting goals of spam protection and non-attributability. We construct efficient protocols that achieve the important security guarantees that DKIM provides, while simultaneously *guaranteeing non-attributability* of stolen email. Further, we show that configurations of our protocols are *practical* for deployment on the Internet today, achieving reasonable efficiency and bandwidth overhead.

### 4.0.1 Key Ideas

There are two main ideas underlying our proposals: *delayed universal forgeability* and *immediate recipient forgeability*.

**Delayed universal forgeability.** This approach ensures that signatures with respect to past emails "expire" after a time delay $\Delta$ and thereafter become forgeable by the general public (i.e., arbitrary outsiders or non-parties). This property ensures that no attribution will be credible after the time delay has elapsed. We call this property *delayed universal forgeability*. As long as $\Delta$ is set larger than the maximum viable time for email latency, the signature will still be convincing to the recipient at the time of receipt, thus maintaining the spam and spoofing-resistance of DKIM.

Signatures that possess delayed universal forgeability retain all the unforgeability properties of a standard signature scheme, until the set time $\Delta$ has passed. Thus in cases where an attacker gains access to email and shows it to a third party within $\Delta$ time after the email was sent, a third party will be convinced of the email's authenticity. Effectively, delayed universal forgeability protects against adversaries that compromise an email account by breaking in and taking a snapshot ("after-the-fact attacks"), but not adversaries that fully control an email account and monitor its email in real time ("real-time attacks"). After-the-fact attacks cover a broad range of realistic attacks, for example, including many data breaches. Next, we discuss how we address real-time attacks.

**Immediate recipient forgeability.** Suppose that the fact of access to a particular client account implies the ability to forge messages from arbitrary other servers *to that recipient only*: that is, the ability to obtain valid DKIM signatures on email content and metadata of one's choice. We call this *immediate recipient forgeability*. Importantly, the recipient constraint ensures the inability to impersonate any other server for the purposes of email addressed to *other* recipients, thus maintaining DKIM's spam and spoofing-resistance. This undermines the credibility of attackers claiming ongoing access to a particular email account and attempting to convince third parties

of the authenticity of emails supposedly sent to (and from) that account — even for real-time attacks, which may publish allegedly-incoming emails immediately as they are received.

Recipient forgeability is weaker than universal forgeability in the following sense: published emails credibly reveal that the attacker has gained access to some users' key material, although not that the email content is authentic. Thus, recipient forgeability is not enough by itself; the two definitions are complementary and incomparable.

**Combining both ideas.** Our protocols attempt to achieve the "best of both worlds," by providing universal forgeability when possible, and falling back on immediate recipient forgeability when necessary. Section 4.2 defines our threat model, discusses its limitations, and formalizes immediate recipient forgeability and delayed universal forgeability.

### 4.0.2 Overview of Solutions

This paper constructs and evaluates two base protocols KeyForge and TimeForge, and two enhanced variants KeyForge$^+$ and TimeForge$^+$ (which consist of the respective base protocol with a modified signing algorithm and one additional sub-protocol). The two base schemes can be seen as two different approaches to building a new type of signature scheme that we introduce: *forward-forgeable signatures (FFS)*.

**Forward-forgeable signatures.** An FFS is a digital signature scheme equipped with a method to selectively disclose signature-invalidating "expiry information" for past signatures without similarly damaging the public key for future signatures. *Succinctness* of FFS is a measure of efficiency of disclosure. We present two constructions of FFS, which are the key building blocks of KeyForge and TimeForge respectively. FFS may be of independent interest as a signature primitive for other applications.

**KeyForge.** Our first proposal, KeyForge (§4.4.1), achieves *delayed universal forgeability* by publishing signing keys after a delay $\Delta$. KeyForge relies on an FFS based on

hierarchical identity-based signatures (HIBS), which achieves logarithmic succinctness. As a result, KeyForge can efficiently distribute forging keys with minimal bandwidth.

**TimeForge.** Our second protocol, TimeForge (§4.4.2), assumes a *publicly verifiable timekeeper (PVTK)* model in which a trusted timekeeper periodically issues publicly verifiable timestamps. In a nutshell, the idea of TimeForge is to substitute each signature on a message $m$ at time $t$ with a succinct zero-knowledge proof of the statement $S(m) \vee T(t + \Delta)$, where: $S(m)$ denotes knowledge of a valid signature by the sender on $m$ and $T(t+\Delta)$ denotes knowledge of a valid timestamp for a time later than $t + \Delta$. Including $T(t + \Delta)$ ensures *delayed universal forgeability.* TimeForge can be described as a forward-forgeable signature scheme in the PVTK model.

**KeyForge$^+$/TimeForge$^+$.** The enhanced protocols (§4.4.4) consist of the respective base protocols with the following modifications: (1) an additional protocol, called *forge-on-request*, that allows parties to request forged emails addressed *only to the requester herself* under limited circumstances; and (2) for multiple-recipient emails, a new signature is produced for each recipient domain (unlike the base protocols and DKIM, which produce one signature per outgoing email).

Among our protocols, KeyForge is the most efficient and would necessitate the least change to existing infrastructure. KeyForge$^+$ and TimeForge$^+$ are alternative approaches showing the feasibility of addressing stronger threat models though at significant overhead (in fact, certain overhead is unavoidable in the stronger threat model; see §4.2). TimeForge could become more practical with advances in the fast-moving area of non-interactive proofs.

**Summary of our Contributions.**

1. We define non-attributability in store-and-forward email systems, and propose two *system designs* — KeyForge (§4.4.1), and TimeForge (§4.4.2) — that achieve this goal.

2. We *implement* KeyForge and TimeForge and evaluate their signing, verification,

and bandwidth costs, and show that KeyForge has acceptable bandwidth and processing overhead for practical deployment (§4.5).

3. We provide formal *definitions* for email non-attributability and prove that our constructions realize them.

4. Of independent interest, we give *provably secure constructions* of a new cryptographic primitive, *succinct forward-forgeable signatures (FFS)*in both the standard and PVTK models (§4.3.2, §4.4.2).

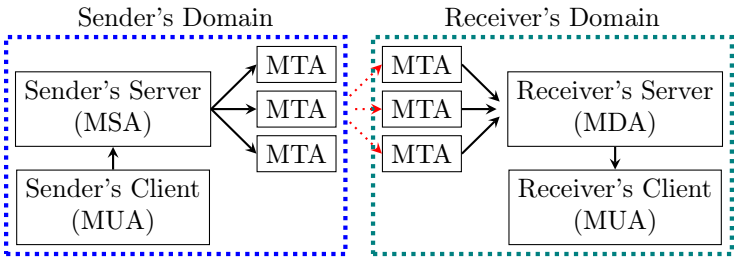



Figure 4-1: Simplified email routing infrastructure

## 4.1 Background on Email

This section introduces basic terminology of mail routing (as defined in RFC 5598 [93]) and describes how email infrastructure necessitates certain system requirements.

As described in Figure 4-1, email uses an asynchronous "store and forward" routing protocol built on top of TCP/IP. Users first establish a relationship with a trusted email service provider, called a Mail Submission Agent (MSA) on the sender side and a Mail Delivery Agent (MDA) on the receiver side. The user's email client is called a Mail User Agent (MUA). Email originates from an MUA, and arrives at the user's trusted MSA. Depending on the system's configuration, the MSA may send the message to intermediary Mail Transfer Agents (MTAs) it trusts. Eventually, as the message leaves the sending server's domain, an MTA will perform a DNS lookup to discover which MTAs are authorized to process messages for the receiving domain, and the email is then sent via SMTP to one of these destination MTAs. After a number of

hops depending on the sending and receiving organizations' infrastructure, the email reaches the receiver's MDA, which is responsible for verifying the message for the receiver's MUA.

### 4.1.1 Email Authentication

The IETF has developed a number of standards that allow domains to sign and verify incoming and outgoing messages. Next, we overview the three that have seen appreciable adoption: DKIM, SPF, and DMARC. Appendix A.2 discusses a fourth, experimental protocol, ARC, and its potential impact.

**DKIM.** DomainKeys Identified Mail (DKIM) is an IETF standard that requires an MSA to sign outgoing email, and an MDA to verify that email by looking up the MSA's public key in the DNS. This procedure is described informally below:

1. *Setup:* The MSA generates a key pair and uploads the public key to the DNS in a TXT record.

2. *Sign:* The MSA adds the location of its public key to the email's metadata (or *header*), as well as additional metadata needed for signature verification, then signs the email and headers with its private key.[1]

3. *Verify:* On receipt, the MDA does a DNS lookup for the MSA's public key, and uses it to verify the signature.

**SPF.** The Sender Policy Framework (SPF) ensures that intermediary MTAs are permitted to send and receive messages as a part of the domain. This solves a somewhat orthogonal problem to DKIM: SPF provides little guarantee that the message has not been modified by an intermediary, but instead provides spoofing protection by limiting what IP addresses are valid accepting MTAs.

[1]This usually includes a hash of the whole message, but the specification does allow for portions of the message to go unsigned. This is not default behavior for most DKIM applications, and has seen limited use in practice.

**DMARC.** An SPF or DKIM failure as a result of a misconfiguration is indistinguishable from a failure due to an attempted message spoofing, and neither DKIM nor SPF provide mechanisms for alerting the sending domain that there has been a problem. DMARC solves this by adding a DNS TXT record specifying to the receiver what it should do in the case of such failures (such as quarantine, reject, or accept the message despite the failure), as well as providing an email address to send aggregated statistics on such failures.

### 4.1.2 DKIM Replacement Constraints

This section overviews a number of demands on email that are not common to many other messaging systems. We find that these requirements make achieving email deniability and security uniquely difficult, and necessitate the new approach we describe in this chapter.

**Indirectness by store and forward.** Email routing is a *store and forward* protocol in which messages are delivered indirectly via multiple hops, and routes, as well as the actual destination addresses, are often not known in advance. To quote the SMTP RFC [185], "[i]t is sometimes difficult for an SMTP server to determine whether or not it is making final delivery since forwarding or other operations may occur after the message is accepted for delivery." Obvious examples of indirectness include *mail forwarding* (in which users configure their MDA to forward email received from an account on one domain to another), and *remailers* (such as mailing lists, that act as MUAs initially); however, there are other, less obvious, places in the ecosystem where this occurs.[2]

For example, many organizations leverage *third-party MTAs* that they do not own as an initial hop between the Internet and the organization's self-hosted MDA/MSA.[3]

[2]Similarly, Mail Retrieval Agents (MRAs) like Getmail [81] behave like MUAs to an MDA, but may forward emails on to an alternate, final MDA. Popular email services like Gmail provide services that download messages from other domains via IMAP.

[3]Third-party MTAs are commonplace. We did an informal survey by scraping DNS MX records for the Alexa top 150k. Surprisingly few, 31,615, have an MX record, and 10,260 use an obvious third-party hosting service (e.g., Google's MTAs), leaving 21,615 that potentially self-host. Of the

These MTAs often provide security benefits to the MDA, such as protection from spam, malicious attachments, or DDoS attacks. While these intermediaries are allowed to quarantine messages or provide flow control to the MDA, under DKIM, they cannot undetectably modify or spoof emails.

In summary, email's indirect, store-and-forward system results in the following constraints: (1) final-destination information (e.g., addresses, keys) may be unknown to the sender, and (2) an MDA may not be certain whether it is the final destination of a message.

**Throughput and scalability.** Email is an any-mesh ecosystem in which any domain owner must be able to set up the appropriate DNS records and interoperate with any other domain's servers. Further, larger domains may sign and verify hundreds to millions of emails per day, and throughput requirements often increase over time. Therefore, beyond good constants on signing and verification time, the service must scale: adding more resources should provide linear or better performance, and scalability in interconnection with other servers is crucial as well.[4]

Such scalability requirements indicate that certain types of overhead that would be trivial in other messaging contexts, (e.g., communication prior to sending a message or per-message round trips between servers), are unlikely to be viable for email. For example, it would be difficult to require the MDA to connect back to the original MSA for every email.

**Long-lived public keys.** One natural approach to short-lived signatures is to leverage correspondingly short-lived keys and publish each secret key at the end of its lifetime, or use short key sizes designed to be able to be brute-forced within the same period (see [73]). This approach has been mentioned in passing outside of the context of email [62]. Unfortunately, too-frequent key rotation entails practical problems that render this tactic unworkable for DKIM. Rotating keys stored in DNS is an often

last category, 31.4% (6,793) are using a confirmed multi-hop third-party MTA. This is likely a conservative estimate, as few servers appear to have matching domain names.

[4]The IETF standard for DMARC [190] states that pre-sending agreements is a poor scalability choice for this reason. See also [290].

manual process that introduces risk of misconfiguration that can cause stability issues, and storing large amounts of key material that must be published, maintained, and shared among several servers is organizationally difficult and increases risk of key theft. DNS results are also often cached, so replacing an individual record is slow and can yield inconsistent results. Finally, it is hard to bound the time for short keys to be broken by all threat actors.

**Incremental deployment.** Given the myriad existing email servers and the need for interoperability, we consider the majority of the email ecosystem to be entrenched. It would be difficult to require substantial changes to mail routing, and it is unrealistic that every actor would promptly switch to a new scheme. Instead, it is far more realistic that DKIM could be replaced by incrementally updating the signing algorithms.

### Resulting System Requirements

The particular constraints of email, described earlier in §4.1.2, rule out many natural approaches to non-attributability, including solutions that might be more feasible in other messaging environments. Since we treat email's indirect, store-and-forward nature as an entrenched property of the infrastructure, realistic proposals for email protocol modifications must not rely on sender use of final-destination information, such as addresses or keys ("Requirement 1" or "R1"). Moreover, due to the store-and-forward and scalability requirements, email protocols should avoid interactive sender-receiver (MDA–MSA) communication whenever possible; in particular, we consider roundtrip sender-receiver communication per email to be inviable ("R2"). Additionally, email protocols must have long-lived public keys ("R3").

Notably, *none* of the following approaches adhere to both the above requirements: interactive zero-knowledge proofs (violate R2); ring signatures (proposed for email non-attributability in [10, 62]) (violate R1); designated-verifier signatures (violate R1); short-lived keys with publication of secret keys after use (violate R3); and — importantly — systems based on deniable authenticated key exchange (DAKE) (which violates R2), such as OTR or Signal [62, 301, 302]. Indeed, both the OTR paper [62,

§6] and a recent DAKE paper [301, §6.6] dedicate a full subsection to discussing the heightened challenges of non-attributability for email as compared to other messaging environments, and note that their proposals are not adequate for email due to its asynchronous, non-interactive, store-and-forward nature. (Table A.1, Appendix A.1 breaks down differences between email and other messaging schemes, and where our solutions fit in, in more detail.)

Finally, we note that the simple approach of relying on MDAs to delete DKIM header information after receipt is flawed *not only* because it fails to address our threat models (§4.2), which require security against *malicious or compromised recipients*, but also because it violates Requirement 1: relying on MDAs for deletion is untenable given that MDAs may not know if they are the final endpoint (and if not, the signatures must be kept for later verification).

**Summary.** A viable non-attributable replacement for DKIM must have: (1) compatibility with indirect, store-and-forward communication (in particular, no reliance on sender knowledge of final destination addresses or keys); (2) no requirement of sender-receiver interaction per email; (3) long-lived public keys; (4) no required behavior for MDAs that depends on whether they are the final destination; (5) little impact on other parts of the email ecosystem; and (6) good systems properties allowing for incremental, scalable deployment.

In addition, a *non-attributable* DKIM replacement must have universal forgeability, and should have recipient forgeability whenever feasible.

## 4.2 Model and Security Definitions

This section presents threat models and formal definitions of email non-attributability.

**Notation.** "PPT" means "probabilistic polynomial time." $|S|$ denotes the size of a set $S$. $[n]$ denotes the set $\{1, \ldots, n\}$ of positive integers up to $n$, and $\mathbb{P}(\cdot)$ denotes powerset. $\approx_c$ denotes computational indistinguishability. $\tau||e$ denotes the result of appending an additional element $e$ to a tuple $\tau$.

### 4.2.1 Model

**Time** We model time in discrete time-steps and assume fairly consistent (say, within 3 mins) local clocks. This is realistic given NTP [69].

**Synchrony** $\hat{\Delta}$ is an upper bound on the time required for email delivery. Our parameter settings depend on $\hat{\Delta}$, and our evaluation sets $\hat{\Delta}$ at 15 minutes (see § 4.4.1).

**DNS** Our model assumes all parties and algorithms have access to DNS and can update their own DNS records.

**Bulletin board** We assume each party has a way to publish persistent, updatable information retrievable by all other parties and algorithms. This could be via DNS or another medium, such as posting on a website. Formally, this can be modeled as a global service $\mathsf{BB}$ that: (1) is initialized with an empty table of key-value pairs; (2) upon receiving a message in $\{\mathsf{write}, \mathsf{append}\} \times \{0,1\}^*$ authenticated with respect to a public key $pk$, respectively (over)writes or appends $x$ to the value (if any) associated with key $pk$; and (3) upon receiving a message of the form $(\mathsf{lookup}, pk)$, responds with the value $x$ associated with $pk$ in the table (if any).

**Publicly verifiable timekeeping service (PVTK)** A PVTK is a global service, initialized with respect to public parameters $pp$ that are publicly known, which maintains a monotonically increasing clock. At any clock time $t$, any party can query the PVTK to obtain a publicly verifiable (w.r.t. $pp$) proof $\pi_t$ that the current PVTK clock time is at least $t$.

In the context of the KeyForge family of protocols, all algorithms are assumed to have the ability to interact with $\mathsf{BB}$. In the context of the TimeForge family of protocols, all algorithms are instead assumed to have the ability to query a PVTK. (To simplify notation, we do not write $A^{\mathsf{BB}}$ or $A^{\mathsf{PVTK}}$ explicitly; but these assumptions will be recalled in the respective sections.)

### 4.2.2 Threat Models

We are concerned with attacks that disclose private communications obtained at the MDA (whether because the MDA is compromised or because it is malicious).

We consider two threat models, defined below. KeyForge and TimeForge achieve security against Threat Model 1, which targets scenarios where attackers may gain access to an email server but are unlikely to maintain access for extended periods. The enhanced protocols KeyForge$^+$ and TimeForge$^+$ achieve security against Threat Model 2, the stronger of the two threat models, which is necessary in settings where attackers' access may likely remain undetected for extended periods (e.g., advanced persistent threats).

**Threat Model 1.** (After-the-fact attacks) *In this model, the recipient is presumed honest at the time of email receipt, but is later compromised by an attacker that takes a snapshot of all stored email content.*

**Threat Model 2.** (Real-time attacks) *In this model, the recipient may be malicious at the time of email receipt, with ongoing and immediate intent to disclose received email content to third parties.*

**Ruling out trivial solutions.** A trivial and uninteresting way to achieve non-attributability, in either threat model, is not to sign emails at all. Of course, this is undesirable as it would undermine the spam- and spoofing-resistance for which DKIM was designed. Providing these guarantees is an implicit requirement throughout this chapter. Moreover, since our threat models consider malicious receiving servers, any non-attributability that relies on receiving-server behavior — such as DKIM header deletion upon receipt — is unsatisfactory.

**Preventing real-time attacks requires interaction.** Any store-and-forward email protocol that both (1) allows recipients to verify the sending domain's identity and (2) is secure against *real-time* attacks (Threat Model 2) must be interactive, as more formally detailed in Claim 1, Appendix A.3. Informally, in the store-and-forward

model, a non-interactive protocol transcript (consisting of a single message from the sender), cannot depend on final-destination recipient information, so any operations (such as verification or forgery) that the verifier can run must also be executable by others. This also relates to the intuitive idea that someone who receives a single message $m$ convincing them of the message's origin must also be able to use $m$ to convince others of the same.

In contrast, security against *after-the-fact* attacks (Threat Model 1) is possible non-interactively, as KeyForge and TimeForge exemplify. KeyForge$^+$ and TimeForge$^+$ augment KeyForge and TimeForge with an interactive (two-message) protocol, which adds significant overhead and complexity to the non-interactive base protocols. Claim 1 shows that this overhead is, in a sense, unavoidable. The overhead of our constructions is furthermore minimal in certain respects: just two rounds of interaction, and the protocols do not *require* interaction on email receipt, but rather, introduce the *possibility* of interaction by an additional protocol (details in §4.4.4).

**What's outside our threat models?** While Threat Model 2 considers powerful real-time adversaries, it too has limits. Definitionally, and unsurprisingly, no deniability is possible against a global passive adversary that can be sure of observing all traffic as it flows over the network. As already mentioned, our threat models are not designed to provide non-attributability against adversaries directly observing email traffic, but rather against those to whom the adversaries might try to pass the stolen emails on.

Our threat models focus on attacks at the receiving server (MDA), because we believe this covers a wide, though not exhaustive, range of attack scenarios of interest. This notably excludes *malicious intermediaries (MTAs)*. Even though our threat models do not focus on MTA-based attacks, our protocols KeyForge$^+$ and TimeForge$^+$ do provide a partial non-attributability guarantee against malicious intermediaries (as discussed in §4.2.3). Nonetheless, malicious intermediaries pose a legitimate concern not fully addressed by this work; achieving stronger non-attributability guarantees against MTAs could be interesting future research.[5]

[5]It is also unclear how effective local MTA-based attacks would be to compromise entire email accounts; such attacks' effectiveness would likely depend on email routing configurations at the servers

Finally, we note that our definitions do not necessarily provide non-attributability against adversaries that can preconfigure the receiving server with custom secure hardware (see also §4.2.3). We consider such attacks outside our threat model: i.e., we assume servers are compromised after physical setup.

We conclude this section with additional context and explanation for our modeling choices.

**Client-server trust.** Email clients rely heavily on their email servers. A malicious email server could easily and undetectably misbehave in many essential functions: e.g., drop incoming emails, modify outgoing emails (since typically, emails are not signed client-side), or falsify content and metadata of incoming emails (since typically, clients do not perform DKIM verification themselves). Since client-server trust is very high in practice, we treat the client and server as a single entity, and relatedly, our threat models do not consider malicious behavior by MSAs that aims to undermine non-attributability of their own clients' emails. (One might also argue such malicious behavior would quickly lose an MSA its clients.)

**Evidence-based credibility.** In a system where credibility is based on reputation rather than evidence — that is, where certain parties' statements are taken on faith, or believed simply because of who they are even without supporting evidence — a "reputable" party with the ability to eavesdrop on the communication channel would be able to undermine non-attributability by keeping traffic logs. Our model assumes mutually distrustful parties: i.e., that no party is taken simply on its word as just described. In other words, credibility in our model is evidence-based and not reputation-based.

---

involved. By entire-account compromise we mean learning all stored emails and/or all real-time emails for a single account over an extended period, as opposed to learning only occasional emails from scattered accounts. Entire-account compromise would be useful to target particular accounts, or to obtain a relatively complete picture of compromised accounts (e.g., for identity theft). In contrast, MDA-based attacks provide a direct way to compromise entire accounts.

**Systemic attributability vs. attributability by choice.** The goal of non-attributability is to empower users to choose whether or not their messages are attributable, to disincentivize email theft and misuse in contrast to attributability-by-default (see §4). We are not concerned with preventing attributability when correspondents desire it: e.g., for business transactions or contracts, correspondents may intentionally sign messages to ensure they are binding. Attribution by journalistic investigation is also outside our threat model: confirmation of selected documents by careful investigation is possible even with handwritten letters, but the current *systemic* attributability facilitates scalable, malicious attribution far beyond the handful of high-profile messages that might be published after arduous manual verification.

### 4.2.3 Defining Non-Attributability

We define email non-attributability as a game involving an email protocol $\mathsf{E} = (\mathsf{Email}, \mathsf{VEmail})$, adversary $\mathcal{A}$, simulator $\mathcal{S}$, and distinguisher $\mathcal{D}$. An email protocol $\mathsf{E}$ is a pair of algorithms, run by the email sender $S$ and recipient(s) $R$ respectively. For an email server $S$ with internal state $s$,[6] $e \leftarrow \mathsf{Email}_s(S, R, m, \mu, t)$ denotes the information (bitstring) transmitted when $S$ sends $R$ an email with message $m$ and metadata $\mu$ at time $t$.[7] The recipient server $R$, upon receiving $e$, runs $\mathsf{VEmail}(e)$, which outputs a single bit indicating whether to accept the email as legitimate or reject it as spoofed.

Intuitively, we require indistinguishability between a legitimate email (denoted by $e \leftarrow \mathsf{Email}_s(S, R, m, \mu, t)$) and a "fake" email that was created without access to the sending server at all: that is, without knowing $s$. To model this, we consider a simulator $\mathcal{S}$ that "aims" to create such an email without $s$, and our security definition

[6] While this definition refers to "internal state $s$" for generality, the state $s$ essentially represents secret key material.

[7] Technically, $e$ may not be the string that $R$ eventually receives, as parties other than the sender (e.g., MTAs) routinely participate in email transmission and may influence the information en route. For simplicity, our notation glosses over this detail and uses $\mathsf{Email}_S(\cdots)$ to refer both to the string $S$ sends and the string $R$ receives. Also, this notation assumes that if an email has multiple recipients, each recipient receives the same information; this is true in the current email system but only some of our protocols. The possibility of different recipients receiving different information is elaborated in §4.4.4, and the notation can easily be tweaked to accommodate this, by treating $R$ as a tuple and having $\mathsf{Email}$ output a tuple of strings. For simplicity, however, we use the single-recipient notation for most of the exposition.

requires $e$ to be distributed indistinguishably from $\mathcal{S}$'s output.

Next, we give two formal definitions of non-attributability. *Recipient non-attributability* (Definition 1) considers a simulator that has access to a particular recipient's email server, and is required to output email from any sender to that recipient. $\Delta$-*universal non-attributability* (Definition 2) is an incomparable definition whose simulator is required to output email from any sender to any recipient while having access to neither the sender's nor the receiver's email server.

In other words, if an adversary publishes an email allegedly authored by an honest party, Definition 1 guarantees that the victim can credibly argue that, granting that the attacker indeed broke into her correspondent's account, the attacker's allegations inherently lack credibility because by the very fact of such access, he could have forged arbitrary emails between them. Definition 2 gives the stronger guarantee that *anyone* can forge past emails after some delay $\Delta$: so after $\Delta$, any allegations are even less credible and the email accounts may not have been compromised at all. The two definitions are complementary and incomparable.

**Definition 1** (Recipient non-attributability)**.** $\mathsf{Email}$ *is* non-attributable for recipients w.r.t. $F$ *if there is a PPT simulator* $\mathcal{S}$ *such that for any sender* $S$ *and recipient* $R$ *(with respective internal states* $s, r$*), for any email message* $m$ *and metadata* $\mu$*,*

$$\mathsf{Email}_s(S, R, m, \mu, t) \approx_c \mathcal{S}_r^F(S, m, \mu) \ ,$$

*where the superscript* $F$ *denotes black-box or query access to an interactive functionality* $F$*, and the subscript* $r$ *denotes that* $\mathcal{S}$ *has access to the recipient server's internal state* $r$*.*[8]

**Definition 2** ($\Delta$-universal non-attributability)**.** *For* $\Delta \in \mathbb{N}$*, an email protocol* $\mathsf{Email}$ *is* $\Delta$-strongly non-attributable *if there is a PPT simulator* $\mathcal{S}$ *such that for any sender*

[8]In fact, our constructions achieve a slightly stronger (i.e., harder to satisfy) definition where $\mathcal{S}$ cannot read $r$, but has only oracle access to signatures by $R$ (produced using key material in $r$). In practice, the latter requirement may be significantly easier to satisfy, as it is achievable by obtaining login access to an email account rather than compromising the server's secrets. However, the definition assumes direct access to $r$ for simplicity.

*$S$ (with internal state $s$) and recipient $R$, for any email message $m$, metadata $\mu$, and timestamp $t$, the following holds at any time $\geq t + \Delta$:*

$$\mathsf{Email}_s(S, R, m, \mu, t) \approx_c \mathcal{S}(S, R, m, \mu, t) \ .$$

Definitions 1 and 2 serve to ensure that no attacker can credibly claim to a third party[9] that he is providing her with authentic emails: the third party is in the role of distinguisher.

Note that Definition 2 is inviable if $\Delta < \hat{\Delta}$. Otherwise, the spam- and spoofing-resistance provided by DKIM would be undermined, since any outsider could use the simulator in real time to send spam email indistinguishable to the recipient from email actually sent by an honest party. Moreover, assuming the essential condition that emails are *not* universally forgeable in real time, Definition 2 implies that the behavior of any $\mathcal{S}$ must differ (distinguishably) between times $\geq t + \Delta$ and times $\leq t$. This is satisfiable only if the view of $\mathcal{S}$ changes between these time intervals: in other words, Definition 2 is satisfiable only if $\mathcal{S}$ gains some new information between these time intervals. In KeyForge and TimeForge, this additional information is made available to $\mathcal{S}$ through the public bulletin board $\mathsf{BB}$ or the PVTK $\mathsf{TK}$, respectively. Absent some time-dependent exogenous functionality like $\mathsf{BB}$ or $\mathsf{TK}$, Definition 2 is (straightforwardly) unsatisfiable.

**Relation to the threat models.** $\Delta$-universal non-attributability achieves non-attributability against after-the-fact attacks (Threat Model 1) for all emails sent and received at least $\Delta$ before the server is compromised.

Combining recipient non-attributability and $\hat{\Delta}$-universal non-attributability (Definitions 1 and 2)yields non-attributability against real-time attacks (Threat Model 2). A real-time attacker with ongoing access to an email server can easily make the fact of his access evident by immediately publishing all emails he sees (within time $\hat{\Delta}$ of receipt), but will be unable to convince third parties of any given email's authenticity

[9]E.g., the general public (if the allegedly stolen emails are released publicly) or a specific interested party (such as a potential buyer or disseminator of the information).

since the fact of his access to the server allows him to forge emails in real time, under Definition 1. For allegedly compromised emails from more than $\hat{\Delta}$ ago, an attacker's credibility is even lower, since for such past timestamps *anyone with internet access* can generate seemingly validly signed emails, even without breaking into any email server at all, under Definition 2.

**Necessity of recipient forgeries.** It may seem a counterintuitive or risky design choice to enable real-time email forgery in any part of the system. If forgery is restricted only to recipients forging emails to themselves, as in our definition, there is no spam/spoofing vulnerability — but given the choice, one might avoid introducing any forging capability at all, in the interest of a simpler and easier-to-analyze system. However, some sort of real-time forging capability by recipients is definitionally necessary to achieve non-attributability against real-time attacks: if the recipient cannot forge in real time, then any third party to whom a recipient server passes emails in real time must be convinced of the emails' authenticity.

**Other inherent model constraints.** A practical consequence of recipient non-attributability is that a recipient $R$'s email server can, *unknown to* $R$, create fraudulent messages that appear to be legitimate emails from any sender to $R$, and deliver them to $R$. As discussed §4.2.2, the current email system necessitates heavy client-server trust. In this context, recipient non-attributability does not meaningfully increase the trust a client places in her email server. For example, email servers in the current system could (and often do) omit DKIM headers when delivering emails to clients: this effectively implies the ability to deliver fake messages.

Also, we note that both definitions allow for strong, persistent attackers to convince others of the very fact that they have ongoing access to a particular email account. The definitions guarantee that even so, such attackers cannot make credible claims about email contents, since they gain the ability to falsify emails by the very fact of their access. That attackers with ongoing access can prove their access is unavoidable since universal forgeability is incompatible with spam resistance for too small $\Delta$, as

discussed above.

**Adversarial secure hardware at recipient.** The requirement of spam- and spoofing-resistance means that any simulator $\mathcal{S}$ satisfying Definition 1 must use the recipient $R$'s secret state $r$: in order to prevent spam, real-time forgery must be limited to messages whose recipient is the forger herself. This suggests that recipient non-attributability would lose meaning in an extreme situation where every use of $r$ can be monitored and attested to, since then an attacker could prove that $\mathcal{S}$ was never invoked on $r$. This might be plausible assuming secure hardware, e.g., by generating and monitoring all uses of $r$ within a secure enclave (as suggested in [155]) — but even then, such an attack would likely only be feasible by the unlikely attacker who has designed her recipient email server with this unlikely configuration from its very setup. We note this possibility for completeness, but *such attacks are outside our threat models*, as mentioned earlier in §4.2.

**Malicious intermediaries and traffic logging.** Although our threat models focus on malicious recipient servers (as discussed earlier in §4.2), Definition 1 actually provides a meaningful, though limited, guarantee against malicious intermediaries (MTAs) as well. If a malicious MTA were to log all traffic and publish it in real time (perhaps even timestamped in a trustworthy way for future reference), in a system with immediate recipient forgeability, observers of the publications would still be unconvinced of: (1) whether any email the MTA claims is genuine (unforged) is really genuine, since the MTA could have *omitted* evidence of forgery, and (2) whether the MTA omitted any genuine emails from its publications.

**Why (sometimes) settle for weaker non-attributability?** KeyForge and TimeForge achieve only non-attributability against after-the-fact attacks, and their enhanced versions KeyForge$^+$ and TimeForge$^+$ are non-attributable against both after-the-fact and real-time attacks. Yet we consider KeyForge to be our main protocol and the most realistic proposal for deployment. In practice, the enhanced protocols' (unavoidable) interactivity and other overhead would often be compelling reasons to prefer

the simpler base protocols except in contexts where addressing real-time attacks (or malicious intermediaries) is of heightened concern.

**Relation to deniability definitions in other contexts.** The cryptographic literature features many works on deniability of signatures and authentication, including (but not at all limited to) [176, 113, 259, 223, 106]. Our constructions could be seen as a practical instantiation of a deniable signature scheme subject to tight systems-based requirements.

Cryptographic deniability definitions tend to come in two flavors: denying communication *content*, or denying having participated in communication at all. Deniable encryption [78] does the former, whereas deniable authentication generally does the latter. Our *recipient-forgeability* is of the former flavor, whereas our *universal forgeability* is of the latter.

## 4.3 Forward-Forgeable Signatures

### 4.3.1 Definition

Definition 3 formalizes *forward-forgeable signatures (FFS)*. They are a new primitive and an essential building block for our proposed protocols. Informally, FFS are signature schemes equipped with a method to selectively "expire" past signatures by releasing *expiry information* that makes them forgeable. In an FFS, each signature is made with respect to a *tag* $\tau$, which is an arbitrary string. Expiry information can be released with respect to any tag or set of tags. In our context, the tag can be thought to be a timestamp: i.e., each email is signed with respect to the current time $\tau$, and at some later time $\tau + \Delta$, the signer may publish expiry information for $\tau$. FFS have *correctness* and *unforgeability* requirements similar to standard signatures, as well as a new requirement, *forgeability on expiry*, that has no analogue in standard signatures.

The *correctness* requirement of FFS is the same as that of standard signatures. The *unforgeability* requirement is modified to include an *expiry oracle*: that is, unforgeability of signatures w.r.t. non-expired tags must hold even in the presence of arbitrary,

adversarially chosen expirations. The *forgeability on expiry* requirement is a feature of FFS that has no analogue in standard signatures.

**Definition 3** (FFS)**.** *A* forward-forgeable signature scheme (FFS) $\Sigma$ *is implicitly parametrized by message space* $\mathcal{M}$ *and tag space* $\mathcal{T}$*, and consists of five algorithms*

$$\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Expire}, \mathsf{Forge})$$

*satisfying the following syntax and requirements.*

Syntax:

- $\mathsf{KeyGen}(1^\kappa)$ *takes as input a security parameter*[10] $1^\kappa$ *and outputs a key pair* $(vk, sk)$*.*
- $\mathsf{Sign}(sk, \tau, m)$ *takes as input a signing key* $sk$*, a tag* $\tau \in \mathcal{T}$*, and a message* $m \in \mathcal{M}$*, and outputs a signature* $\sigma$*.*
- $\mathsf{Verify}(vk, \tau, m, \sigma)$ *takes as input a verification key* $vk$*, a tag* $\tau \in \mathcal{T}$*, a message* $m \in \mathcal{M}$*, and a signature* $\sigma$*, and outputs a single bit indicating whether or not* $\sigma$ *is a valid signature with respect to* $vk$*,* $m$*, and* $\tau$*.*
- $\mathsf{Expire}(sk, T)$ *takes as input a signing key* $sk$ *and a tag set* $T \subseteq \mathcal{T}$*, and outputs expiry info* $\eta$*.*
- $\mathsf{Forge}(\eta, \tau, m)$ *takes as input expiry info* $\eta$*, a tag* $\tau \in \mathcal{T}$*, and a message* $m \in \mathcal{M}$*, and outputs signature* $\sigma$*.*

Required properties:

1. <u>Correctness:</u> *For all* $m \in \mathcal{M}, \tau \in \mathcal{T}$*, there is a negligible function* $\varepsilon$ *such that for all* $\kappa$*,*

$$\Pr\left[\begin{array}{ll} (vk, sk) \leftarrow \mathsf{KeyGen}(1^\kappa) & \\ \sigma \leftarrow \mathsf{Sign}(sk, \tau, m) & : \quad b = 1 \\ b \leftarrow \mathsf{Verify}\,(vk, \tau, m, \sigma) & \end{array}\right] \geq 1 - \varepsilon(\kappa)\ .$$

[10] Technically, all five algorithms take $1^\kappa$ as an input, and $\mathcal{M}$ and $\mathcal{T}$ may be parametrized by $\kappa$. For brevity, we leave this implicit except in $\mathsf{KeyGen}$.

2. Unforgeability: *For any PPT $\mathcal{A}$, there is a negligible function $\varepsilon$ such that for all $\kappa \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{ll} (vk, sk) \leftarrow \mathsf{KeyGen}(1^\kappa) & \\ (\tau, m, \sigma) \leftarrow \mathcal{A}^{\mathsf{S}_{sk}, \mathsf{E}_{sk}}(vk) & : \quad b = b' = 1 \\ b \leftarrow \mathsf{Verify}\,(vk, \tau, m, \sigma) & \\ b' = \tau \notin Q'_{\mathsf{E}} \wedge (\tau, m) \notin Q_{\mathsf{S}} & \end{array}\right] \leq \varepsilon(\kappa)\ ,$$

*where $\mathsf{S}_{sk}$ and $\mathsf{E}_{sk}$ respectively denote oracles $\mathsf{Sign}(sk, \cdot, \cdot)$ and $\mathsf{Expire}(sk, \cdot)$, $Q_{\mathsf{S}}$ and $Q_{\mathsf{E}}$ denote the sets of queries made by $\mathcal{A}$ to the respective oracles, and $Q'_{\mathsf{E}} = \bigcup_{T \in Q_{\mathsf{E}}} T$.*

3. Forgeability on expiry: *For all $m \in \mathcal{M}, T \subseteq \mathcal{T}$, for any $\tau \in T$, for any "distinguisher" algorithm $\mathcal{D}$, there is a negligible function $\varepsilon$ such that for all $\kappa$,*

$$\Pr\left[\begin{array}{ll} (vk, sk) \leftarrow \mathsf{KeyGen}(1^\kappa) & \\ \sigma_0 \leftarrow \mathsf{Sign}(sk, \tau, m) & \\ \eta \leftarrow \mathsf{Expire}(sk, T) & : \quad b = b' \\ \sigma_1 \leftarrow \mathsf{Forge}(\eta, \tau, m) & \\ b \leftarrow \{0, 1\} & \\ b' \leftarrow \mathcal{D}(\sigma_b, \eta) & \end{array}\right] \leq 1/2 + \varepsilon(\kappa)\ .$$

*That is, $\mathcal{D}$ must not be able to distinguish whether a signature was produced using* Sign *or* Forge*, even in the presence of the expiry information $\eta$.*

The *forgeability on expiry* property requires computational indistinguishability between a signature produced using Sign and a signature produced using Forge on valid expiry information. In particular, when combined with the *correctness* property, this implies that any such signature produced with Forge must appear to be a valid signature, i.e., cause Verify to output 1.

**FFS in the publicly verifiable timekeeper model** Recall the *publicly verifiable timekeeper* (PVTK) model, In this model, expiration may occur "automatically": signers need not publish additional expiry information for signatures to become

forgeable after a delay. Thus, the algorithm Expire is unnecessary, and Forge need not take input $\eta$. In §4.4.2, we construct FFS in the PVTK model.

**Difference with forward-secure signatures** Both FFS and FSS yield a system of short-lived secret keys all corresponding to one long-lived public key. However, the definitions differ in two main ways, described below and depicted in Figure 4-2.

1. Forward-*secure* signatures require that past keys cannot be computed from future keys, whereas forward-*forgeable* signatures require that future keys cannot be computed from past keys.

2. Forward-*secure* signatures are designed to prevent compromise of past signatures by compromising a later secret key. All FSS secret keys are short-lived and each secret key must be derivable based solely on the previous short-lived secret key. Forward-*forgeable* signatures, in contrast, may have persistent "master secret key" material used to generate each short-lived key.

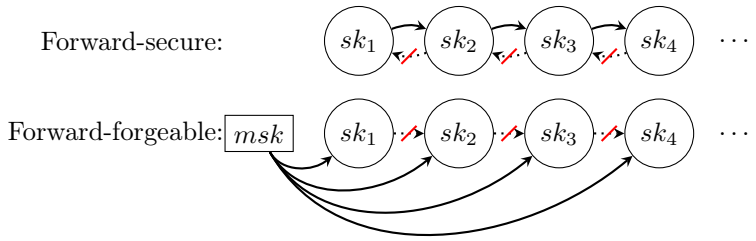



Figure 4-2: Forward-secure vs. forward-forgeable signatures

**Succinctness** The *succinctness* of an FFS is a measure of the efficiency of disclosure in terms of the size of expiry info per tag expired. Concretely, in our application, succinctness measures how expiry info scales as more non-attributable emails are exchanged over time. KeyForge uses a construction of FFS based on hierarchical identity-based signatures (§4.3.2), which achieves logarithmic succinctness.

**Definition 4.** *Let $z : \mathbb{N} \to \mathbb{N}$. Let $S \subset \mathbb{P}(\mathcal{T})$ be a set of sets of tags. A forward-forgeable signature scheme $\Sigma$ is $(S, z)$-*succinct *if for any $T \in S$, there is a negligible*

*function $\varepsilon$ such that for all $\kappa$,*

$$\Pr_{(vk,sk)\leftarrow\mathsf{KeyGen}(1^\kappa)}\left[\left|\mathsf{Expire}(sk,T)\right| \leq z(|T|)\right] \geq 1-\varepsilon(\kappa)\ .$$

*Remark* 1. Definition 4 is a worst-case definition: it guarantees the size of expiry information with overwhelming probability. In certain applications, an average-case definition may be appropriate instead, i.e., defining succinctness by bounding the size on *expectation*. We use a worst-case definition since it is stronger than an average-case definition, and our construction achieves it.

### 4.3.2 FFS Construction from (Hierachical) IBS

We first outline a simple FFS construction BasicFFS based on identity-based signatures (IBS) [273], as a stepping stone to our main construction from hierarchical IBS (HIBS). The next paragraph assumes familiarity with standard IBS terminology; readers unfamiliar with IBS may skip to the main construction which is explained formally with explicit syntax definitions.

Let tags in the FFS correspond to identities in the IBS. BasicFFS.KeyGen outputs IBS master keys. The BasicFFS signing and verification algorithms for tag $\tau$ respectively invoke the IBS signing and verification algorithms for identity $\tau$. BasicFFS.Expire outputs the secret key for each input tag $\tau \in T$, and BasicFFS.Forge uses the appropriate secret key from the expiry information to invoke the IBS signing algorithm. This simple solution has linear succinctness. By leveraging hierarchical IBS (HIBS), our main construction achieves logarithmic succinctness, as described next.

**Definition 5.** *A* hierarchical identity-based signature scheme HIBS *is parametrized by message space $\mathcal{M}$ and identity space $\mathcal{I} = \{\mathcal{I}_\ell\}_{\ell\in\mathbb{N}}$, and consists of four algorithms* $\mathsf{HIBS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *with the following syntax.*

- $\mathsf{Setup}(1^\kappa)$ *takes as input a security parameter*[11] *and outputs a master key pair* $(mvk, msk)$.

[11] Technically, all four algorithms take $1^\kappa$ as an input, and $\mathcal{M}$ and $\mathcal{I}$ may be parametrized by $\kappa$. For brevity, we leave this implicit except in Setup.

- $\mathsf{KeyGen}(sk_{\vec{id}}, id)$ *takes as input a secret key* $sk_{\vec{id}}$ *for a tuple of identities* $\vec{id} = (id_1, \ldots, id_\ell) \in \mathcal{I}_1 \times \cdots \times \mathcal{I}_\ell$ *and an additional identity* $id \in \mathcal{I}_{\ell+1}$ *and outputs a signing key* $sk_{\vec{id}'}$ *where* $\vec{id}' = (id_1, \ldots, id_\ell, id)$*. The tuple may be empty (i.e.,* $\ell = 0$*): in this case,* $sk_{()} = msk$*.*

- $\mathsf{Sign}(sk_{\vec{id}}, m)$ *takes as input a signing key* $sk_{\vec{id}}$ *and a message* $m \in \mathcal{M}$*, and outputs a signature* $\sigma$*.*

- $\mathsf{Verify}(mvk, \vec{id}, m, \sigma)$ *takes as input master verification key* $mvk$*, tuple of identities* $\vec{id}$*, message* $m \in \mathcal{M}$*, and signature* $\sigma$*, and outputs a single bit indicating whether or not* $\sigma$ *is a valid signature with respect to* $mvk$*,* $\vec{id}$*, and* $m$*.*

*A* depth-$L$ HIBS *is a HIBS where the maximum length of identity tuples is* $L$*, i.e., the identity space is* $\mathcal{I} = \{\mathcal{I}_\ell\}_{\ell \in [L]}$*.*

**Definition 6.** *For an identity space* $\mathcal{I} = \{\mathcal{I}_\ell\}_{\ell \in \mathbb{N}}$*, we say* $\vec{id}$ *is a* level-$\ell$ identity *if* $\vec{id} \in \mathcal{I}_1 \times \cdots \times \mathcal{I}_\ell$*. For any* $\ell' > \ell$*, let* $\vec{id}$ *be a level-*$\ell$ *identity and* $\vec{id}'$ *be a level-*$\ell'$ *identity. We say that* $\vec{id}'$ *is a* sub-identity *of* $\vec{id}$ *if* $\vec{id}$ *is a prefix of* $\vec{id}'$*. If moreover* $\ell' = \ell + 1$*, we say* $\vec{id}'$ *is a* immediate sub-identity *of* $\vec{id}$*.*

**Deriving subkeys** Given a master secret key of a HIBS, it is possible to derive secret keys corresponding to level-$\ell$ identities for any $\ell$, by running $\mathsf{KeyGen}$ $\ell$ times. By a similar procedure, given any secret key corresponding to a level-$\ell$ identity $\vec{id}$, it is possible to derive any "subkeys" thereof, i.e., secret keys for sub-identities of $\vec{id}$. For our construction, it is useful to name this (simple) procedure: we define $\mathsf{HIBS.KeyGen}^\star$ in Algorithm 1. We write the randomness $\rho_1, \ldots, \rho_\ell$ of $\mathsf{HIBS.KeyGen}^\star$ explicitly.

**Algorithm 1** $\mathsf{HIBS.KeyGen}^\star$

**Input:** $sk, \ell, \vec{id} = (id_1, \ldots, id_{\ell'})$ ▷ *Require:* $\ell \leq \ell'$
**Randomness:** $\rho_1, \ldots, \rho_{\ell'}$
  **for** $j = \ell + 1, \ldots, \ell'$ **do**
    $sk \leftarrow \mathsf{HIBS.KeyGen}(sk, id_j; \rho_j)$
  **return** $sk$

**HIBS security requirements** Definition 5 gives only syntax and not security requirements. For a formal security definition, see, e.g., [141]. Informally, an HIBS must satisfy the following:

- *Correctness:* For any identity tuple $\vec{id}$, an honestly produced signature w.r.t $\vec{id}$ must verify as valid w.r.t. $\vec{id}$.
- *Unforgeability:* For any PPT adversary $\mathcal{A}$ with access to a $\mathsf{KeyGen}(msk, \cdot, \cdot)$ oracle, the probability that $\mathcal{A}$ outputs a valid signature w.r.t. an identity $\vec{id} \notin Q$ must be negligible, where $Q$ is the set of all sub-identities of identities $\mathcal{A}$ has queried to the oracle.

**Algorithm 2** Compress

**Input:** $\mathcal{I} = \{\mathcal{I}_\ell\}_{\ell \in [L]}, T \subseteq \mathcal{I}_1 \times \cdots \times \mathcal{I}_\ell$
  // *Remove redundant sub-identities*
  **for all** $\tau \in T$ **do**
    **if** $\exists \tau' \in T$ s.t. $\tau'$ is a prefix of $\tau$ **then**
      $T = T \setminus \{\tau\}$
  // *Replace identities with prefix identities where possible*
  **for** $\ell = L-1, \ldots, 1$ **do**
    **for all** $\vec{\tau} = (\tau_1, \ldots, \tau_\ell) \in \mathcal{I}_1 \times \cdots \times \mathcal{I}_\ell$ **do**
      // *$X$ represents all level-$(\ell+1)$ sub-identities of $\vec{\tau}$*
      $X = \{(\tau_1, \ldots, \tau_\ell, \tau')\}_{\tau' \in \mathcal{I}_{\ell+1}}$
      **if** $X \subseteq T$ **then**
        $T = T \setminus X$
        $T = T \cup \{(\tau_1, \ldots, \tau_\ell)\}$
  **return** $sk$

**Succinctly representing expiry information** Given any set $T$ of tuples of identities, the simplest way to make signatures with respect to $T$ forgeable would be to release the secret key corresponding to each $\vec{id} \in T$, much as in $\mathsf{BasicFFS}$:

$$\eta = \left\{ sk_{\vec{id}} = \mathsf{HIBS.KeyGen}^\star(msk, 0, \vec{id}) \right\}_{\vec{id} \in T} . \tag{4.1}$$

However, leveraging the hierarchical nature of HIBS, $\eta$ can often be represented more succinctly than (4.1). Based on the fact that Algorithm 1 allows the derivation of

any subkey, we make two optimizations. First, before computing (4.1), we delete from $T$ any $\vec{id} \in T$ that is a sub-identity of some $\vec{id}' \in T$. Secondly, if there is any $\vec{id}' = (id_1, \ldots, id_\ell) \in \mathcal{I}_1 \times \cdots \times \mathcal{I}_\ell$ such that every immediate subkey of $\vec{id}'$ is in $T$, i.e.,

$$\forall id_{\ell+1} \in \mathcal{I}_{\ell+1},\ (id_1, \ldots, id_\ell, id_{\ell+1}) \in T\ ,$$

then all sub-identities of $id'$ can be removed from $T$ and replaced by $id'$ before computing (4.1). Such replacement is permissible only when *every* possible subkey of $id'$ is derivable from $T$: otherwise, adding $id'$ to $T$ would implicate additional subkeys beyond those originally in $T$.

These two optimizations yield an algorithm Compress, which takes as input a set of identity tuples $T$, and outputs a (weakly) smaller set of identity tuples $T'$ such that knowledge of the secret keys corresponding to $T'$ enables computing valid signatures with respect to exactly the identity tuples in $T$. HIBS security guarantees that even given $T'$, signatures for identity tuples not in $T$ remain unforgeable. Next, we describe how Compress works using a tree-based representation of identity tuples Algorithm 2 fully specifies Compress.

**Tree representation** It is convenient to think of identity tuples represented graphically in a tree. A node at depth $\ell$ represents a tuple of $\ell$ identities (considering the root node to be at depth 0). The set of all depth-$\ell$ nodes corresponds to the set of all $\ell$-tuples of identities. The branching factor at level $\ell$ is $|\mathcal{I}_{\ell+1}|$. Given a secret key for a particular node (i.e., identity tuple), the secret keys of all its descendant nodes are easily computable using HIBS.KeyGen$^\star$. (The secret key for the root node is the master secret key.) In this language, Compress simply takes a set $T$ of nodes and returns the smallest set $T'$ of nodes such that (1) all nodes in $T$ are descendants of some node in $T'$ and (2) no node not in $T$ is a descendant of any node in $T'$. Figure 4-3 gives an illustration of the Compress algorithm on a small example tree.

Our construction of FFS based on HIBS follows. It makes use of Algorithms 1 and 2, which were just defined.

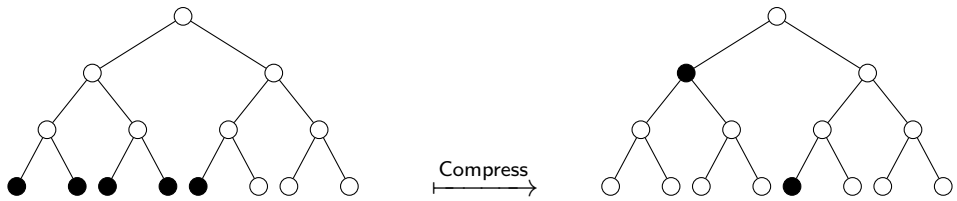



Figure 4-3: Example application of Compress

**Construction 1.** *Let* HIBS *be a depth-L HIBS*[12] *with message space* $\mathcal{M}$ *and identity space* $\mathcal{I} = \{\mathcal{I}_\ell\}_{\ell \in [L]}$. *Let* $\mathcal{O}$ *be a random oracle,*[13] *and for any tuple* $\vec{\tau} = (\tau_1, \ldots, \tau_\ell)$, *let* $\vec{\mathcal{O}}(\vec{\tau}) = (\mathcal{O}(\tau_1), \ldots, \mathcal{O}(\tau_\ell))$. *For* $\ell \in [L]$, *define* $\mathcal{T}_\ell = \mathcal{I}_1 \times \cdots \times \mathcal{I}_\ell$. *We construct a FFS* $\Sigma$ *with message space* $\mathcal{M}$ *and tag space* $\mathcal{T} = \bigcup_{\ell \in [L]} \mathcal{T}_\ell$, *as follows.*

- $\Sigma.\mathsf{KeyGen}(1^\kappa)$: *output* $(vk, sk) \leftarrow \mathsf{HIBS.Setup}(1^\kappa)$.
- $\Sigma.\mathsf{Sign}(sk, \vec{\tau} = (\tau_1, \ldots, \tau_\ell), m)$: *let*

$$sk_{\vec{\tau}} = \mathsf{HIBS.KeyGen}^\star(sk, 0, \vec{\tau}; \vec{\mathcal{O}}(\vec{\tau}))$$

  *and output* $\sigma \leftarrow \mathsf{HIBS.Sign}(sk_{\vec{\tau}}, m)$.

- $\Sigma.\mathsf{Verify}(vk, \vec{\tau}, m, \sigma)$: *output* $b \leftarrow \mathsf{HIBS.Verify}(vk, \vec{\tau}, m, \sigma)$.
- $\Sigma.\mathsf{Expire}(sk, T)$: *let* $T' = \mathsf{Compress}(\mathcal{I}, T)$; *output*

$$\eta = \left\{ (\vec{\tau}, sk_{\vec{\tau}}) : sk_{\vec{\tau}} = \mathsf{HIBS.KeyGen}^\star(sk, 0, \vec{\tau}; \vec{\mathcal{O}}(\vec{\tau})) \right\}_{\tau \in T'} .$$

- $\Sigma.\mathsf{Forge}(\eta, \tau, m)$: *if there exists* $sk_{\tau'}$ *such that* $(\tau', sk_{\tau'}) \in \eta$ *and* $\tau'$ *is a prefix of* $\tau$, *let* $\ell$ *be the length of* $\tau'$, *let*

$$sk_{\vec{\tau}} = \mathsf{HIBS.KeyGen}^\star(sk_{\tau'}, \ell, \vec{\tau}; \vec{\mathcal{O}}(\vec{\tau}))$$

  *and output* $\sigma \leftarrow \mathsf{HIBS.Sign}(sk_{\vec{\tau}}, m)$; *otherwise, output* $\perp$.

[12] The depth need not be finite, but we consider finite $L$ for simplicity.

[13] The construction is presented in the random oracle model for simplicity, but does not *require* a random oracle: the random oracle can be replaced straightforwardly by a pseudorandom function (PRF) where the PRF key is made part of the HIBS secret key.

**Theorem 1.** *If* HIBS *is a secure HIBS, Construction 1 instantiated with* HIBS *is a forward-forgeable signature scheme.*

*Proof.* Correctness and unforgeability of Construction 1 follow directly from correctness and unforgeability of the underlying HIBS. The FFS requirement of *forgeability on expiry* moreover follows from the correctness requirement of the HIBS: the Forge algorithm of Construction 1 invokes HIBS.Sign using a secret key $sk_{\vec{\tau}}$ which is guaranteed (by construction of HIBS.KeyGen$^\star$) to be the secret key corresponding to identity tuple $\tau$. The validity of HIBS.Sign invoked on a valid secret key corresponding to identity tuple $\tau$ is guaranteed by the correctness of the HIBS. □

**Lemma 1** (Logarithmic succinctness)**.** *Let* HIBS *be a depth-$L$ HIBS with message space $\mathcal{M}$ and identity space $\mathcal{I} = \{\mathcal{I}_\ell\}_{\ell\in[L]}$. For each $\ell \in L$, let $\preceq_\ell$ be a total order on $\mathcal{I}_\ell$. Let $\mathcal{T}_L = \mathcal{I}_1 \times \cdots \times \mathcal{I}_L$. For $i \in |\mathcal{T}_L|$, let $\tau_i$ denote the $i$th element of $\mathcal{T}_L$ in the lexicographic order induced by $\{\preceq_\ell\}_{\ell\in L}$. Construction 1 instantiated with* HIBS *is $(S, 2z)$-succinct and also $(S_1, z)$-succinct, where*

$$S = \left\{\{\tau_i\}_{j'\le i\le j} : j, j' \in [|\mathcal{T}_L|]\right\} \quad \textit{and} \quad S_1 = \left\{\{\tau_i\}_{1\le i\le j} : j \in [|\mathcal{T}_L|]\right\} .$$

*and $z(\cdot) = B \cdot \log_B(\cdot)$ where $B = \max_{\ell\in L}\{|\mathcal{I}_\ell|\}$. We assume $B$ is constant.*

*Proof.* Fix any $j, j' \in [|\mathcal{T}_L|]$ and any set $T = \{\tau_i\}_{j\le i\le j'}$. By the definition of succinctness, it suffices to show that the output of Compress on $T$, is a set of nodes of size at most $2B \cdot \log(|T|)$.

For any identity tuple $\iota \in \mathcal{I}$, let $\mathrm{Sub}_\iota$ be the set of all level-$L$ identities of which $\iota$ is a prefix. We say that $T$ *covers* $\iota$ if $\mathrm{Sub}_\iota \subseteq T$. Let $\mathrm{Cover}_T$ be the set of all identities covered by $T$. We say $T$ *subsumes* an identity $\iota$ if $\iota$ is a descendant of some $\iota' \in \mathrm{Cover}_T$ such that $\iota' \neq \iota$. By construction of Algorithm 2 (Compress), any identity *subsumed* by $T$ will not be in the output set of Compress($T$) (specifically, it will be removed in the innermost for-loop of Algorithm 2).

For any $\ell \in [L]$, consider any consecutive sequence of $s$ level-$\ell$ identities covered by $T$. By definition of lexicographic ordering, there are fewer than $2B$ level-$\ell$ identities

in the sequence that are not subsumed by $T$ (these identities will be at the beginning and/or end of the sequence). Moreover, if the sequence begins at the smallest level-$\ell$ identity in the lexicographic order, then there are fewer than $B$ identities in the sequence that are not subsumed by $T$ (these identities will be at the end of the sequence).

Thus, for each $\ell \in [L]$, there are fewer than $2B$ level-$\ell$ identities in the output set of $\mathsf{Compress}(T)$. Moreover, $L \leq \log_B(|\mathcal{T}_L|) \leq \log_B(T)$. Therefore, the output set size is at most $2B \log_B(T)$. Moreover, if $T \in S_1$, then the output set size is at most $B \log_B(T)$. □

**Discussion of alternative approaches** As discussed above, forward-*secure* signatures (FSS) and FFS are different primitives with distinct requirements. One could build a FFS from a FSS by computing a long list of secret keys and then using them *in backwards order*. Using techniques of [175, 90], a sequence of keys could moreover be stored with logarithmic storage and computation to access a key. However, this optimization is only designed for contiguous sequences of keys; HIBS-based schemes allow for some succinct non-sequential key release and thus support more nuanced tag structures. Still, for certain applications, e.g., postquantum sequential key release, an FFS based on a FSS such as XMSS [66] could be useful.

The requirements of FFS also have some similarity to *timed authentication*. The TESLA timed authentication protocol [244, 245] considers releasing authentication (MAC) keys following a delay after sending the payload, in the broadcast authentication context. Such delayed verification is untenable for email for several reasons, even beyond the inconvenience of waiting 15 minutes for email delivery. Email's store-and-forward nature (see §4.1.2) means multiple MTAs may need to verify emails before forwarding (e.g., for spam filtering): if the first MTA waits to verify before forwarding, the next MTA will be unable to verify because the delay has rendered the authentication forgeable. Also, the inability to discard incoming spam before a time delay may increase denial-of-service vulnerability, especially for smaller email providers.

## 4.4 Our Protocol Proposals

Sections 4.4.1 and 4.4.2 respectively describe our two proposed systems KeyForge and TimeForge.

### 4.4.1 KeyForge

KeyForge consists of two components: (1) replace the digital signature scheme used in DKIM with a succinct FFS; and (2) email servers periodically publish expiry information. In this section, we assume all algorithms have access to a global publication mechanism or bulletin board (as noted in §4.2.1).

**FFS configuration for KeyForge.** Figure 4-4 illustrates KeyForge's key hierarchy. KeyForge is based on an $L$-level tag structure, corresponding to identity space $\mathcal{I} = \{\mathcal{I}_\ell\}_{\ell \in [L]}$ where the level-$L$ identities represent 15-minute time chunks spanning a 2-year period. We use the following intuitive 4-level configuration for ease of exposition, but as discussed in §4.5, it is preferable for efficiency to keep $|\mathcal{I}_\ell|$ equal for all $\ell \in [L]$.

| | |
|---|---|
| $\mathcal{I}_1 = \{1, 2\}$ | representing a 2-year time span |
| $\mathcal{I}_2 = \{1, \ldots, 12\}$ | representing months in a year |
| $\mathcal{I}_3 = \{1, \ldots, 31\}$ | representing days in a month |
| $\mathcal{I}_4 = \{1, \ldots, 96\}$ | representing 15-minute chunks of a day |

A tag $\tau = (\mathsf{y}, \mathsf{m}, \mathsf{d}, \mathsf{c}) \in \mathcal{I}_1 \times \mathcal{I}_2 \times \mathcal{I}_3 \times \mathcal{I}_4$ corresponds to a 15-minute chunk of time. The 15-minute chunks are contiguous, consecutive, and disjoint, so that any given timestamp is contained in exactly one chunk. $\tau(t)$ denotes the unique 4-tuple tag $(\mathsf{y}, \mathsf{m}, \mathsf{d}, \mathsf{c})$ that represents the chunk of time containing a timestamp $t$, and $t \sqsubset \tau$ denotes that $\tau$ represents a chunk of time containing timestamp $t$.

KeyForge requires each signature at time $t$ to be with respect to a tag (timestamp) $t + \hat{\Delta}$. The tag is sent in the email's header, and used for verification at the receiving server. Algorithm 3 specifies KeyForge's signing and verification (key generation is

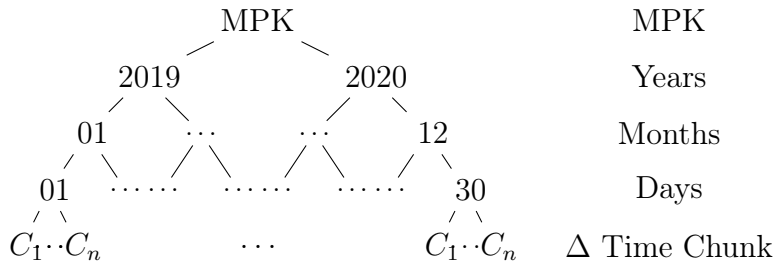



Figure 4-4: KeyForge Hierarchy Layout

identical to that of the underlying FFS).

**Algorithm 3** KeyForge.Sign and KeyForge.Verify

```
t = CurrentTime()
function KeyForge.Sign(sk, m, Δ)
    return (τ(t + Δ), σ ← FFS.Sign(sk, τ(t + Δ), m))
function KeyForge.Verify(vk, m, τ, σ)
    return t ⊏ τ AND FFS.Verify(vk, τ, m, σ)
```

Then we build an email protocol $\mathsf{E_{KF}} = (\mathsf{Email}, \mathsf{VEmail})$ as follows. Let $(vk, sk)$ be the key pair of a sending server. $\mathsf{Email}_{sk}(S, R, m, \mu, t)$ outputs $(\zeta, \tau_\zeta, \sigma_\zeta)$ where $\zeta = (S, R, m, \mu)$, $(\tau_\zeta, \sigma_\zeta) \leftarrow \mathsf{KeyForge.Sign}_t(sk, \zeta, \hat{\Delta})$, and the subscript $t$ denotes an execution of KeyForge.Sign at time $t$. $\mathsf{VEmail}(\zeta, \tau_\zeta, \sigma_\zeta)$ runs $\mathsf{KeyForge.Verify}(vk, \zeta, \tau_\zeta, \sigma_\zeta)$ and outputs the result (where the recipient obtains $vk$ by looking up $S$'s key in DNS).

**Efficient tree regeneration from private keys.** A key feature of our FFS construction is that the private keys from children (e.g., day-keys) are easy to generate from parent keys (e.g., the MSK). This is not implied by the definition of HIBS,[14] and is essential for succinct expiry of entire portions of the tree (e.g., a year) by disseminating a single key. Further, regeneration can enhance security and availability: to limit key exposure, organizations could store the MSK in an HSM disconnected from the Internet, and keep only a child key pair in the MSA, thereby mitigating damage in case of compromise and allowing recovery from failure.

[14]The definition of HIBS is compatible with this property, but does not require it. Constructions typically have randomized subkey generation processes so do not have reproducible child keys.

**Where to publish expiry information?** Regeneration allows KeyForge to have succinct expiry information; the number of private keys necessary to represent all expired chunks depends on the tree's structure (see §4.5), but amounts to less than 4 KB for reasonable configurations. In contrast, the analogous construction based on non-hierarchical IBS would have expiry information growing linearly throughout a (two-year) master-key lifespan, resulting in megabytes of expiry information.

Small expiry information means ease of distribution. While our implementation uses a simple public-facing webserver, one could imagine posting via DNS TXT records, public blockchains, or in outgoing email headers. Slow but permanent techniques (e.g., a blockchain) for keys higher up in the hierarchy (e.g., a year) could ensure that such keys are permanently available.

**When to publish expiry information?** KeyForge requires email servers to publish expiry information at regular intervals. A natural option is to publish expiry information every 15 minutes; to publish the expiry information corresponding to each chunk $\mathsf{c}$ at the end of the time period that $\mathsf{c}$ represents.

Publishing every 15 minutes yields the finest granularity of expiry possible under the basic four-level tag structure. Based on a server's preference, it could release information at longer intervals (e.g., days) or shorter ones. In case of an attack, an adversary would be able to convince third parties of the authenticity of all emails in the current interval (e.g., the current day), so risk aversion prefers shorter intervals.

Server misconfiguration and clock skew may cause minor clock discrepancies between the MSA and MDA. To account for this, we delay publishing "expired" keys by 5 minutes. Although in practice most emails are received very quickly, the SMTP RFC [185] has a very lax give-up time of 4 days. To get a rough idea of how quickly emails tend to be delivered, we computed the time differences from the first Received header to the last in the Podesta email corpus [321],[15] and found that, of the 48,246 messages with parseable Received timestamps, over 99% (47,349) took less than 12 minutes.

[15]Beyond the irony, we chose the Podesta email corpus as it was distributed intact with attachments, and thus arguably more representative of a realistic user's email distribution than other public datasets.

While expiry time is a configurable parameter of KeyForge (e.g., by administrators), keeping it short is advisable to minimize time until universal forgeability. We leave a detailed study of email delivery times in practice to future work, while noting that such a study might support considerably reducing our conservative 15 minutes, and/or tailoring our approach to specific delay-prone situations. For example, delays are often caused by expected receiving-server outages (e.g., for server updates), which might be resolved by using a DMARC-like DNS record to signal to the sender to hold messages until later. Anti-spam techniques such as greylisting can delay email by 15 minutes more than usual; to address this, we can add 17 minutes' leeway when first sending to a new domain.

We do not fully detail remediation procedures for timeouts, but note that similar authentication failures happen under DKIM and are commonly resolved via feedback loops such as Authentication Failure Reporting [128]. Shortening our expiry time is tricky given potentially adversarial routing delays: providing TCP-like flow control would be systematically possible, but we should also account for malicious MTAs trying to prolong messages' unforgeability. A hard-cutoff maximum would likely be advisable.

**Why 15-minute chunks?** The time period associated with each leaf node is the maximum granularity of expiry information release. $\hat{\Delta}$ is a lower bound on chunk size: since $\hat{\Delta}$ represents email delivery time, publishing expiry information more often does not make sense.

**Why a 2-year public key lifetime?** Rotating keys is good practice; for operational reasons, the Messaging, Malware, and Mobile Anti-Abuse Working Group (M3AAWG) recommends DKIM key rotation every 6 months [192]. However, recognizing that, realistically, DKIM keys often last more than 6 months, our evaluation assumes a 2-year period.

**How many levels?** The optimal $L$ depends on a trade-off between computation time and expiry succinctness; see §4.5.

**Flexible expiry policies** The basic tag structure described above is customizable: e.g., an extra level $\mathcal{I}_\ell$ might represent an email's "sensitivity," allowing sensitive emails to expire faster. Alternatively, one might want certain emails to expire more slowly or never (e.g., bank/employer emails or contracts). In KeyForge, sensitivity can be expressed as the desired delay until expiry. KeyForge is highly configurable: after the first four levels, different email servers' policies need not be consistent. Verification refers only to the first four levels of the tag (when checking $t \sqsubset \tau$), so a sending server can add more levels beyond the basic four without sacrificing compatibility, to match its desired expiry policy.

### 4.4.2 TimeForge

KeyForge's main limitation is that it requires signers to continuously release key material. Wide distribution can pose a practical challenge; users must depend on their provider to perform this task reliably. Unreliable distribution would limit a system's realistic deniability.

TimeForge takes a different approach that eliminates reliance on follow-up action by signers. TimeForge leverages a *publicly verifiable timekeeping service* (PVTK), as defined in §4.2.1. In this section, all algorithms are assumed to have access to a common PVTK.

The intuition behind TimeForge is straightforward. Let $M$ be an email message sent at time period $t$. The sender first signs each message using a standard SUF-CMA signature scheme to produce a signature $\sigma$. She then authenticates the message, not directly using $\sigma$, but rather using a witness indistinguishable and non-interactive proof-of-knowledge (PoK) of the (informal) statement:

I know a valid sender signature $\sigma$ on $M$

OR

I know a valid PVTK proof $\pi_{t+d}$, for some $d \geq \Delta$.

Assuming a trustworthy PVTK service, this proof authenticates the message during any time period prior to $t + \Delta$. Once a PVTK proof $\pi_{t+\Delta}$ becomes public, the PoK

becomes trivial for any party to generate. Witness indistinguishability ensures that a signer's valid proof is indistinguishable from a "forgery" later computed using a revealed PVTK proof.

**Publicly verifiable timekeeping.** A PVTK scheme comprises three algorithms.

- $\mathsf{TK.Setup}(1^\lambda)$ takes a security parameter $\lambda$ and outputs a set of public parameters $params$ and a trapdoor $sk$.
- $\mathsf{TK.Prove}(sk, t)$ takes as input $sk$ and the current time epoch $t$, and outputs a proof $\pi_t$.
- $\mathsf{TK.Verify}(params, t, \pi_t)$ on input $params$, a time period $t$, and the proof $\pi_t$, outputs whether $\pi_t$ is valid.

*Correctness and Security.* Correctness is straightforward. $\Delta$-*PVTK security* requires that an adversary with a PVTK oracle (which provides proofs for arbitrary time periods $t$) must not be able to produce a valid proof for some time period $t_{\mathsf{max}} + \Delta$ (except with negligible probability) where $t_{\mathsf{max}}$ is the largest oracle query, and $\Delta > 0$ is a constant parameter.

**Realizing a PVTK service.** A simple PVTK system can be constructed using a single server that maintains a clock, and periodically signs the current time using an SUF-CMA signature (our implementation does this).

While conceptually simple, deploying this solution at scale is likely to be costly, and may suffer denial-of-service and network-based attacks. A better approach might construct a PVTK from *existing* Internet services: next, we outline several proposals.

**OCSP servers.** The Online Certificate Status Protocol, in its "stapling" configuration [246] allows TLS servers to obtain a standalone, signed, and timestamped certificate validity message from a Certificate Authority (CA); this can be viewed as an organic implementation of a PVTK server. To avoid reliance on a single CA, users can define the proof $\pi_t$ to comprise *multiple* valid staples, *e.g.,* one from any $k$ out

of $N$ chosen CAs. These parameters, as well as the CA identities, can be selected as part of the setup algorithm.

**CT and randomness beacons.** The Certificate Transparency protocol consists of a centrally-managed and publicly verifiable log for recording the issuance of certificates [196]. While CT is not intended as a timestamping protocol, each CT log entry is signed by the log operator (*e.g.,* Google), contains a timestamp, and may be re-purposed to implement a centralized PVTK service. Similarly, NIST operates a randomness beacon [231] that signs and distributes timestamps. While any single centralized service may be unreliable or subject to attack, a (fault-tolerant) combination of these extant services can be used to construct a "composite" PVTK system.

**Proof of work blockchains.** A number of cryptocurrencies use *proof of work* blockchains to construct an ordered transaction ledger [222, 324] which generates new blocks at a rate using intentionally chosen parameters. These ledgers can be used as a form of PVTK system, in which *params* comprises some initial block header $B_s$, and $\pi_t$ comprises a list of block headers $\{B_{s+1}, \ldots, B_t\}$ drawn from the blockchain. Although this approach does not produce an exact timekeeping service (block intervals are probabilistic), nor does it guarantee cryptographic unforgeability (as chains can be forged given control of a substantial fraction of the network's hash power), this is likely not a problem for the short intervals used in TimeForge.

**VDFs and puzzles.** Cryptographic "puzzles" are mathematical problems that require a known (or statistically predictable) number of computational operations to solve. Examples include cryptocurrency proof of work systems and timelock encryption schemes [257]. A related primitive, the *Verifiable Delay Function* [58, 248] creates a *sequential* puzzle that requires a precisely-known amount of work to solve, and allows the solver to produce a proof of the solution's correctness. While puzzles and VDFs do not directly allow for the creation of a PVTK system, they enable a related primitive: at time $t$ a sender may generate a puzzle challenge $M$ (*e.g.,* the contents of an email message) such that a proof $\pi_{t+\Delta}$ can be found by applying a computational process to

$M$ in expected time approximately $\Delta$.

**A basic TimeForge signature scheme.** The TimeForge scheme consists of four algorithms: $\mathsf{TF.Keygen}$, $\mathsf{TF.Sign}$ and $\mathsf{TF.Verify}$, and $\mathsf{TF.Forge}$. We assume a PVTK scheme with parameters $params$ and an SUF-CMA signing algorithm $\mathsf{Sig}$.

- $\mathsf{TF.Keygen}(1^\lambda, params)$. Run $\mathsf{Sig.Keygen}(1^\lambda)$ to generate $(pk, sk)$ and output $PK = (pk, params)$, and $SK = sk$.

- $\mathsf{TF.Sign}(PK, SK, M, t, \Delta)$. Parse $PK = (pk, params)$. On input a message $M$ and a time period $t$, compute $\sigma \leftarrow \mathsf{Sig.Sign}(SK, M\|t\|\Delta)$ and the following witness-indistinguishable (WI) non-interactive PoK:[16]

$$\Pi = \mathsf{NIPoK}\{(\sigma, s, \pi) : \mathsf{Sig.Verify}(pk, \sigma, M\|t\|\Delta) = 1 \ \vee \\ (\mathsf{TK.Verify}(params, \pi, s) = 1 \ \wedge \ s \geq t + \Delta)\}$$

  Note that the prover can produce this proof using $(\sigma, \perp, \perp)$ as the witness. Output $\sigma_{\mathsf{tf}} = (\Pi, t, \Delta)$.

- $\mathsf{TF.Verify}(PK, M, \sigma_{\mathsf{tf}})$. Parse $PK = (pk, params)$ and $\sigma_{\mathsf{tf}} = (\Pi, t, \Delta)$, verify the proof $\Pi$ with respect to the public values $t, \Delta, pk, M$, and output the verification result.

$\mathsf{TF.Forge}$ takes as input a PVTK proof $\pi_s$ for some time period $s \geq t + \Delta$.

- $\mathsf{TF.Forge}(PK, M, t, s, \Delta, \pi_s)$. parse $PK = (pk, params)$ and compute the NIPoK $\Pi$ described in the $\mathsf{TF.Sign}$ algorithm, using the witness $(\perp, s, \pi_s)$. Output $\sigma_{\mathsf{tf}} = (\Pi, t, \Delta)$.

*Defining Security.* Security for TimeForge is defined according to the following experiment. This experiment can be considered a variant of (weak) $\mathsf{UF\text{-}CMA}$ security definition for a signature scheme: an attacker must attempt to forge a TimeForge

[16] Here we use Camenisch-Stadler notation, where the witness values are in parentheses () and any remaining values are assumed to be public.

proof over a message $M$ that she has not previously queried to a signing oracle. To assist in this, the attacker is given access to not one, but two oracles. The first is a signing oracle for the TimeForge signature scheme, and produces valid signatures for tuples of the form $(M', t', \Delta')$. The main difference from the standard $\mathsf{UF\text{-}CMA}$ experiment is the existence of a second oracle that models the PVTK service. To model this, the attacker additionally obtains PVTK parameters *params* at the start of the experiment, and may repeatedly query the PVTK oracle on chosen epoch numbers $s$ to obtain PVTK proofs of the form $\pi_s$. Let $s_{\mathsf{max}}$ be the largest time period queried to the PVTK oracle at the conclusion of the experiment. We say the attacker *wins* iff she outputs a message $M$ and valid TimeForge proof $\sigma_{\mathsf{tf}} = (\Pi, t, \Delta)$ such that $s_{\mathsf{max}} < t + \Delta$. where $(M, t, \Delta)$ was not queried to the signing oracle. We say that a TimeForge scheme is unforgeable under *chosen timestamp attacks* if $\forall$ p.p.t. attackers $\mathcal{A}$, the adversary has at most a negligible advantage in succeeding at the above experiment.

*Remark* 2. The definition above does not prevent forgeries that are "outside the expiration period." Specifically, an intermediary can intercept a message embedding $(M, t, \Delta)$ where $t + \Delta$ is in the future, and author a new message $M', t', \Delta'$ where $t + \Delta$ has already been proved by the PVTK oracle. This is explicitly allowed by TimeForge; indeed, it is a goal of the system.

**Theorem 2.** *If (1) the PVTK service uses an* $\mathsf{SUF\text{-}CMA}$ *signature scheme, (2) the WI proof system is sound (extractable), and (3) the underlying signature scheme used by TimeForge is* $\mathsf{SUF\text{-}CMA}$*, then the basic TimeForge scheme is secure under chosen timestamp attacks.*

*Proof.* Our proof is by contradiction. Let $\mathcal{A}$ be an attacker that succeeds with non-negligible advantage in the chosen timestamp experiment. We construct a pair of algorithms $\mathcal{B}_1, \mathcal{B}_2$ such that one of the two algorithms (respectively) succeeds with non-negligible advantage in the $\mathsf{SUF\text{-}CMA}$ game against (1) the signature scheme $\mathsf{Sig}$, or (2) the underlying PVTK signature scheme. We now describe the operation of each algorithm.

*An attack on the signature scheme* Sig. In this strategy we construct $\mathcal{B}_1$, which conducts the SUF-CMA experiment for the underlying signature scheme Sig. $\mathcal{B}_1$ first obtains a public key $pk$ from the SUF-CMA challenger. It next uses the PVTK signature scheme's key generation algorithm to produce a keypair $(params, sk_{\mathsf{PVTK}})$ for the PVTK service and sends $PK = (pk, params)$ to $\mathcal{A}$.

Each time $\mathcal{A}$ queries the PVTK oracle on some timestamp $s$, $\mathcal{B}_1$ implements TK.Prove by using $sk_{\mathsf{PVTK}}$ to sign $s$ and return the signature $\pi_s$. Whenever $\mathcal{A}$ queries the TimeForge signing oracle on some tuple $(M, t, \Delta)$, $\mathcal{B}_1$ first queries the SUF-CMA signing oracle to obtain a signature $\sigma$ on $M\|t\|\Delta$. It then constructs a TimeForge signature by constructing the proof described in the TF.Sign algorithm using $\sigma$ as the satisfying witness, and returns $\sigma_{\mathsf{tf}}$ to $\mathcal{A}$.

When $\mathcal{A}$ outputs a pair $(M^*, \sigma^*_{\mathsf{tf}})$ that satisfies the win conditions of the experiment, $\mathcal{B}_1$ parses $\sigma^*_{\mathsf{tf}}$ to obtain $(\Pi^*, t^*, \Delta^*)$ and runs the extractor on $\Pi^*$ to obtain the witness $(\sigma^*, s^*, \pi^*)$. (If the extractor fails, $\mathcal{B}_1$ aborts.) If $\sigma^*$ is a valid signature on $M^*\|t\|\Delta$, it outputs the pair $(\sigma^*, M^*)$ as an SUF-CMA forgery.[17]

*An attack on the PVTK scheme.* In this strategy we construct $\mathcal{B}_2$, which conducts the SUF-CMA experiment against the PVTK signature scheme. This algorithm proceeds as in Strategy 1, except that here we set *params* to be the public key obtained from the SUF-CMA challenger, and generate the keypair $(pk, sk) = \mathsf{Sig.Keygen}(1^\lambda)$. Queries to the PVTK oracle are answered by forwarding $s$ to the SUF-CMA oracle and returning the resulting signature as $\pi_s$, and queries to the TimeForge oracle are answered honestly by running TF.Sign with $sk$ as an input. As in the previous strategy, if $\mathcal{A}$ succeeds in the experiment we run the extractor to obtain $\sigma^*, s^*, \pi^*$. Then $\mathcal{B}_2$ verifies that $\pi^*$ is a valid signature on $s^*$ and, if so, outputs $(\pi^*, s^*)$ as a forgery for the SUF-CMA experiment.

**Analysis.** $\mathcal{A}$'s view is distributed identically when it interacts with $\mathcal{B}_1$ or $\mathcal{B}_2$, so $\mathcal{A}$'s advantage must be identical w.r.t. $\mathcal{B}_1$ and $\mathcal{B}_2$. By soundness, the WI knowledge extractor fails with probability at most negligible in the security parameter. Thus

[17] By the restrictions on $\mathcal{A}$, $M^*$ must represent a message that has not previously been queried to the SUF-CMA oracle.

both adversaries will abort with at most negligible probability due to extraction error.

It remains only to show that at least one of the two algorithms above must succeed with non-negligible advantage in the SUF-CMA experiment. This is true because the attacker's output $(\Pi^*, t^*, \Delta^*)$ and the extracted witness $(\sigma^*, s^*, \pi^*)$ must satisfy the conditions that (1) for all PVTK queries $s$ made during the experiment, $s < t^* + \Delta^*$ (by the requirements of the experiment), (2) the message $M^* \| t \| \Delta$ has not been queried to the SUF-CMA signing oracle (by the requirements of the experiment), and (3) the following conditions are true (by the soundness of the proof system):

$$\begin{aligned} \mathsf{Sig.Verify}(pk, \sigma^*, M^*) = 1 \ \vee \ & (\mathsf{TK.Verify}(params, \pi^*, s^*) \\ = 1 \ \wedge \ & s^* \geq t^* + \Delta^*) \end{aligned}$$

If $\mathcal{A}$ succeeds in the TimeForge experiment with non-negligible advantage, then the extracted witness must contain a valid signature $\sigma^*$ on a message $M^* \| t \| \Delta$ not queried to the signing oracle, **or** it must contain a valid signature $\pi^*$ on some epoch $s^*$ not queried to the PVTK oracle. Hence, one of $\mathcal{B}_1$ or $\mathcal{B}_2$ succeeds with non-negligible advantage. □

### 4.4.3 Realizing the TimeForge proof system

TimeForge can be realized using a variety of WI and ZK proof systems, combined with efficient SUF-CMA signature schemes. For example, a number of pairing-based signature schemes [39, 77, 40] admit efficient proofs of knowledge of a signature using simple Schnorr-style proofs [10]. More recent proving systems, e.g., Bulletproofs [67] and zkSNARKs (e.g., [239, 153], admit succinct proofs of statements involving arbitrary arithmetic circuits and discrete-log relationships. Using the latter schemes ensures short proofs, in the hundreds of bytes, in some cases with a small, constant verification cost. Thus, even complex PVTK proofs such as block header sequences, can potentially be reduced to a succinct TimeForge signature.

**A concrete implementation.** For our basic implementation, which signs a timestamp, we considered several proof systems. For the relatively simple proof statement used in this scheme, Bulletproofs are not appropriate for two reasons: (1) the proof sizes that result exceed 1000 bytes, and (2) these proofs do not natively support efficient signatures. zkSNARKs produce bandwidth-efficient signatures, but at a significant cost due to the need to generate a trusted setup embedding the signature verification circuit. Based on these considerations, we propose and evaluate one concrete implementation based on Schnorr-style proving techniques, made non-interactive using the Fiat-Shamir heuristic. Our approach implements TimeForge using a dedicated server that produces (weak) Boneh-Boyen signatures [59] over the current time period $t$, which is encoded as an integer in $\mathbb{Z}_q$. Let $g_1, g_2$ be generators of a pair of bilinear groups $\mathbb{G}_1, \mathbb{G}_2$ of order $q$. Briefly, a Boneh-Boyen signature on a time period $t$ comprises a single group element $\sigma = g_1^{1/x+t}$, where $x$ represents the signing key, and the server's public key is $g_2^x$. Verification is conducted by checking the following pairing equality: $e(g_1, g_2) = e(\sigma, g_2^x g_2^t)$.

Our proposed TimeForge proof of knowledge requires the following components. First, the prover to provides a Pedersen commitment $B$ to the current time period $T_{\mathsf{current}}$ using randomness $r$. The proof also reveals the (alleged) true signing time period $T_{\mathsf{signing}}$ in cleartext (in case it is different) and attaches $\delta$. Using these values, the prover employs the homomorphic property of Pedersen commitments to derive an implicit commitment $C = g_1^{\gamma = T_{\mathsf{current}} - T_{\mathsf{signing}} - \delta} h^{r'}$, and then uses a range proof to prove that it knows a value $\gamma$ that is in the range $[1, 2^{32}]$. We use a range proof due to Camenisch, Chaabouni, and shelat [76]. Alternatively, this proof could be implemented using a Bulletproof, due to Bootle *et al.* [67].

In addition to this commitment proof and range proof, we provide two separate Schnorr-style proofs in an "OR" construction:

1. *A standard Schnorr signature on the message.* This comprises an interactive proof of knowledge of a value $sk \in \mathbb{Z}_q$ such that $PK = g^{sk}$, flattened into a signature of knowledge on the signed message, using the Fiat-Shamir heuristic. (This represents the genuine signer's signature on the message.)

2. *A proof of knowledge of a Boneh-Boyen signature on the TimeForge time period $T_{\mathsf{current}}$, signed using the TimeForge server secret key.* For this construction we use a interactive zero-knowledge protocol given by Boneh, Boyen and Shacham [60, Protocol 1], flattened using the same Fiat-Shamir hash function.

### 4.4.4 KeyForge$^+$ and TimeForge$^+$

KeyForge$^+$ (resp. TimeForge$^+$) consists of KeyForge (resp. TimeForge) with two modifications: a *forge-on-request protocol* and *per-recipient-domain signatures*, described next.

**1. Forge-on-request protocol.** We add a protocol $\mathsf{F}$ (detailed in Algorithm 6) by which an email server $S$ accepts real-time requests for specified email content to be sent to the requester (and nobody else). We write $A^{\mathsf{F}}$ to denote that an algorithm $A$ has access to email forgeries via $\mathsf{F}$. The forge-on-request protocol ensures that all users have the capability to forge emails to themselves in real time, directly achieving *immediate recipient forgeability*. The requirement that the recipient be the requester is crucial: each requester is enabled to forge emails *only to herself*.

The requester's email server attests to the requesting client's identity (similarly to DKIM). We note that a malicious server could unauthorizedly sign requests for any client account it controls. This is outside our threat model, and such behavior is equally possible under DKIM (see also "Client-server trust" under §4.2.2): that is, today's email ecosystem already relies on servers to attest honestly to their clients' identities, and allows servers to spam their own clients (a behavior that might not keep them many clients).

The scheme as described so far already works for single-recipient emails, but can be problematic for multi-recipient[18] emails due to spam/spoofing attacks between co-recipients: as long as all recipients of an email receive exactly the same information, an attacker can always request a forged email addressed to a group including herself, and quickly forward it to trick the others into treating the email as legitimate. To

[18]Here, "recipients" means any recipients whether via `to`, `cc`, or `bcc`.

remedy this, we require one more change to KeyForge.

**2. Per-recipient-domain signatures.** In DKIM, KeyForge, and TimeForge, the sending server signs each outgoing email once. In KeyForge$^+$ and TimeForge$^+$, instead, the sending server signs each outgoing email *once per recipient domain.* That is, the sending server produces a signature $\sigma_D = \mathsf{Sign}(sk, (D, m))$ for each recipient domain $D$,where $sk$ is the signing key and $m$ is the email information that the sending server would have signed under DKIM (or KeyForge or TimeForge). The sending server sends each recipient domain $D$ the email and *only* the signature $\sigma_D$.

Per-recipient-domain signatures prevent attackers from using the forge-on-request protocol to send spam/spoofing emails to co-recipients on forged emails. Adida *et al.* [9] previously proposed per-recipient signatures in a very similar context.

Note that a forge-on-request protocol achieves a stronger guarantee than the use of ring signatures (i.e., signing with respect to both the sending and receiving servers' public keys). A forge-on-request protocol enables any recipient with the ability to send mail from an email server to forge mail from any sender to herself. The ring signature approach enables her to do this only if she has the ability to sign fraudulent mail with the receiving server's secret key.

We define the email protocols $\mathsf{E}_{\mathsf{KF}+}$ and $\mathsf{E}_{\mathsf{TF}+}$ accordingly:

$\mathsf{E}_{\mathsf{KF}+}.\mathsf{Email}(S, (R_1, \ldots, R_N), m, \mu, t)$ outputs $(e_1, \ldots, e_N)$ where $e_i \leftarrow \mathsf{E}_{\mathsf{KF}}.\mathsf{Email}(S, R_i, m, \mu, t)$; and $\mathsf{E}_{\mathsf{KF}+}.\mathsf{VEmail}$ is just as in $\mathsf{E}_{\mathsf{KF}}$. $\mathsf{E}_{\mathsf{TF}+}$ is defined analogously, w.r.t. $\mathsf{E}_{\mathsf{TF}}$.

**Theorem 3.** $\mathsf{E}_{\mathsf{KF}}$ *and* $\mathsf{E}_{\mathsf{KF}+}$ *are* $\hat{\Delta}$*-universally non-attributable (Definition 2). Assuming email servers adopt the forge-on-request protocol* $\mathsf{F}$*,* $\mathsf{E}_{\mathsf{KF}+}$ *is further non-attributable for recipients (Definition 1).*

*Proof.* Follows directly from Lemmata 2 and 3. □

**Lemma 2.** $\mathsf{E}_{\mathsf{KF}+}$ *is non-attributable for recipients w.r.t.* $\mathsf{F}$ *(Definition 1), where* $\mathsf{F}$ *is the forge-on-request functionality defined in Algorithm 6.*

*Proof.* Let $\mathsf{E}_{\mathsf{KF+}} = (\mathsf{Email}, \mathsf{VEmail})$. Recall from Definition 1 that we must show that there is a PPT simulator $\mathcal{S}$ such that for any sender $S$ and recipient $R$, for any email message $m$ and metadata $\mu$,

$$\mathsf{Email}_{sk}(S, R, m, \mu, t) \approx_c \mathcal{S}^R(S, m, \mu) \ , \tag{4.2}$$

where $sk$ is the (master) secret key of $S$, $t$ is the time at which $\mathcal{S}$ is invoked, and the superscript $R$ denotes that $\mathcal{S}$ has access to the recipient's email server. We construct $\mathcal{S}$ in Algorithm 4.

**Algorithm 4** Simulator $\mathcal{S}$ for recipient non-attributability

**Input:** $S, m, \mu$
  $t = \mathsf{CurrentTime}()$
  **send** forge request $(m, \mu)$ to $S$
  **receive** answer $\{e_0, e_1\}$
  **parse** $e_0, e_1$ as emails w.r.t. tags $\tau_0, \tau_1$ respectively
  **if** $\tau(t) = \tau_0$ **then return** $e_0$
  **else if** $\tau(t) = \tau_1$ **then return** $e_1$

By definition of $\hat{\Delta}$, we know $S$ received the request at some time $t' \leq t + \hat{\Delta}$. Thus, by construction of KeyForge, the emails $e, e'$ must be signed with respect to the tags $\tau(t' - \hat{\Delta}), \tau(t')$ (say, respectively). It follows that $\tau(t) = \tau(t' - \hat{\Delta})$ or $\tau(t) = \tau(t')$. Therefore, at least one of the if-conditions in Algorithm 4 must be satisfied, and $\mathcal{S}$ always produces an output. By construction of the if-statements and the forge-on-request protocol, the output of $\mathcal{S}$ is an email signed by $S$ for a tag corresponding to timestamp $t$, as (4.2) requires.Indeed, we achieve equality of distributions (not just indistinguishability). □

**Lemma 3.** $\mathsf{E}_{\mathsf{KF+}}$ *is* $\hat{\Delta}$*-universally non-attributable (Def. 2).*

*Proof.* Let $\mathsf{E}_{\mathsf{KF+}} = (\mathsf{Email}, \mathsf{VEmail})$. Recall from Definition 2 that we must show that there is a PPT simulator $\mathcal{S}$ such that for any sender $S$ (with secret key $sk$) and recipient $R$, for any email message $m$, metadata $\mu$, and timestamp $t$, the following

holds at any time $\geq t + \hat{\Delta}$:

$$\mathsf{Email}_{sk}(S, R, m, \mu, t) \approx_c \mathcal{S}(S, R, m, \mu, t) \ . \tag{4.3}$$

Let $\mathsf{Email}^*$ be identical to $\mathsf{Email}$ except that whenever $\mathsf{Email}_{sk}$ invokes $\mathsf{KeyForge.Sign}(sk, \cdot)$, $\mathsf{Email}^*_{sk^*}$ instead invokes $\mathsf{HIBS.Sign}(sk, \cdot)$. Next, we construct $\mathcal{S}$ in Algorithm 5.

**Algorithm 5** Simulator $\mathcal{S}$ for $\hat{\Delta}$-strong non-attributability

**Input:** $S, R, m, \mu, t$
  **retrieve** published expiry information $\eta$ for $S$
  **for all** $(\vec{\tau}, sk_{\vec{\tau}}) \in \eta$ **do**
    **if** $t \sqsubset \vec{\tau}$ **then**
      $sk^* \leftarrow \mathsf{HIBS.KeyGen}^\star(sk_{\vec{\tau}}, \ell, \tau(t); \vec{\mathcal{O}}(\vec{\tau}))$
      **return** $\mathsf{KeyForge}^*_{sk^*}(S, R, m, \mu, t)$

Since $\mathcal{S}$ is invoked at time $\geq t + \hat{\Delta}$, and KeyForge prescribes publication of expiry information at the end of each chunk of duration $\hat{\Delta}$, the expiry information $\eta$ retrieved by $\mathcal{S}$ includes expiry information with respect to time $t$. Therefore, the if-condition in Algorithm 5 is satisfied for at least one element of $\eta$.[19]

Recall that $\mathsf{KeyForge.Sign}$ invokes $\mathsf{FFS.Sign}$, which invokes $\mathsf{HIBS.Sign}$. By definition, if $t \sqsubset \vec{\tau}$ and $\eta$ is expiry information with respect to $sk$, then $sk^*$ as computed in Algorithm 5 is the same key used to invoke $\mathsf{HIBS.Sign}$ (within $\mathsf{FFS.Sign}$, which is within $\mathsf{KeyForge.Sign}$) at time $t$. Therefore, the output distributions of $\mathsf{Email}^*_{sk^*}$ and $\mathsf{Email}_{sk}$ are identical. It follows that $\mathcal{S}$ satisfies (4.3). □

**Theorem 4.** *Assuming a PVTK,* $\mathsf{E_{TF}}$ *and* $\mathsf{E_{TF+}}$ *are* $\hat{\Delta}$*-universally non-attributable and* $\mathsf{E_{TF+}}$ *is further non-attributable for recipients.*

**On the efficiency of KeyForge$^+$/TimeForge$^+$** Per-recipient-domain signatures add sender-side overhead compared to single-signature schemes like DKIM, KeyForge, or TimeForge: the sending server must compute one signature per recipient domain per email, whereas the receiving server verifies just one signature, just like before.

[19] In fact, it will be satisfied for exactly one element of $\eta$, by construction of $\mathsf{Compress}$ which ensures that no timestamp is represented by more than one element.

While this additional computation is unlikely to be prohibitive given the efficiency of signing, it must be taken into account when evaluating KeyForge$^+$ and TimeForge$^+$'s efficiency, as discussed further in Section 4.5.

Implementing forge-on-request and per-recipient-domain signatures would entail more complexity and significant changes to the existing email infrastructure, than the base protocols. While immediate recipient forgeability is desirable for added protection against real-time attacks (see Threat Model 2), KeyForge is a more realistic candidate for near-term deployment as it is realizable with lighter-weight changes to the existing system: namely, replacing DKIM's signature scheme, and unilateral server publication of small amounts of data.

**Notation** $\mathsf{Email}_s(S, R, m, \mu, t)$ is as defined in §4.2.3, additionally taking into account that signatures in KeyForge$^+$ and TimeForge$^+$ are per recipient domain. $\mathsf{FReq}$ denotes a special message to betoken forge requests. For an email address $a$, let $a$.dom denote its domain.

## 4.5 Implementation and Evaluation

We implemented prototypes of KeyForge and TimeForge and integrated them into Postfix, a common MDA/MSA. The entire project consists of roughly 2,000 lines of Go, C, and Python, and is available open source.[23] We performed all benchmarks on a 2017 MacBook Pro, 15-inch, with an Intel 4-core 3.1GHz processor and 16GB of RAM. We use the RELIC toolkit's [29] implementation of a BN-254 curve. This configuration conservatively yields keys with a 110-bit security level [30], which is on par with the standard 2048-bit RSA. We chose RELIC due to its support for many pairing friendly curves and low overhead.

We evaluate two versions of KeyForge instantiated with different HIBS schemes: (1) KeyForge$_B$, which uses Gentry-Silverberg's "BasicHIDE" bilinear map based scheme

[20] We assume client-server communication is authenticated.
[21] I.e., if the request is for a forgery from another address in the requester's own domain.
[22] $\mathsf{Sign}$ denotes the signing algorithm of any secure signature scheme.
[23] https://github.com/keyforgery/KeyForge

**Algorithm 6** Forge-on-request protocol $\mathsf{F}$

**Requester (client)**
To request an email with message $m$ and metadata $\mu$ from `alice@foo.com`:

- Send $(\mathsf{FReq}, m, \mu, \texttt{alice@foo.com})$ to client's (i.e., its own) email server.

**Email server** (say, `bar.com`, with secret key $s$)
On receiving request $(\mathsf{FReq}, m, \mu, a)$ from own client `bob`:[20]

- If $a$.dom = `bar.com`:[21] Let $t$ be the current time. Deliver $e$ to `bob`, where $e \leftarrow \mathsf{Email}_s(a, \texttt{bob@bar.com}, m, \mu, t)$.
- Else: Let $\sigma \leftarrow \mathsf{Sign}(\mathsf{FReq}, m, \mu, a, \texttt{bob})$.[22] Send $(\mathsf{FReq}, m, \mu, a, \texttt{bob}, \sigma)$ to server $a$.dom.

On receiving request $(\mathsf{FReq}, m, \mu, a, b, \sigma)$ from server $b$.dom:

- $v \leftarrow \mathsf{Verify}(vk, (\mathsf{FReq}, m, \mu, a, b), \sigma)$, where $pk$ is $b$.dom's public key in DNS.
- If $v = 0$: Do not respond.
- Else (i.e., $v = 1$): Let $t$ be the current time. Send $e, e'$ to $b$.dom, where $e \leftarrow \mathsf{Email}_s(a, b, m, \mu, t)$ and $e' \leftarrow \mathsf{Email}_s(a, b, m, \mu, t - \hat{\Delta})$.

[141] using a BN254 curve and (2) KeyForge$_C$, which uses certificate chains on public keys using non-identity-based signatures, instantiated with Ed25519.[24] We also implemented a prototype of TimeForge which is less efficient; it is intended as a proof of concept whose practicality will improve with advances in the underlying proof primitives (an active area of research). The two KeyForge implementations share the following bandwidth optimization.

**KeyForge bandwidth optimization.** HIBS schemes tend to have relatively large signatures. In KeyForge$_B$, a signature must include public parameters for each node on the path to the current chunk. A public parameter in this configuration is 65B, yielding a bandwidth of 260B for a four-level Year/Month/Day/Chunk tree, resulting in a total of 293B per signature. KeyForge$_C$ similarly requires an Ed25519 signature between each node in the hierarchy, and has total signature size of 448B (four 64B path signatures, four 32B public keys, and the message signature). We optimize bandwidth by precomputing all path parameters except for the last chunk and store them in the DNS, along with the MPK. When verifying from a new server, KeyForge performs a DNS lookup and caches the result at a cost of 2-3KB per month (see Table 4.1).

Two components, the keyserver and mail filter, are shared between all implementations. They are described next.

**Mail Filter.** The filter ensures that sent emails are properly formatted, verifies incoming emails, and communicates the results to the MDA/MTA. The filter works by intercepting SMTP requests, adding necessary metadata to outgoing email headers, and requesting cryptographic operations from the keyserver. When sending a message, the filter attaches an expiry time (and other verification information) to the email's header, hashes the metadata and message content, forwards the hash to the keyserver to sign, and finally adds this signature to the header. On receipt, the filter confirms that the signature's hash matches the message and metadata, and forwards the signature, sending domain, and expiry timestamp to the keyserver for verification. If verification

[24] The certificate-based approach has been attributed to folklore.

| Monthly KeyForge$_B$ | KeyForge$_B$ $\sigma$ | Monthly KeyForge$_C$ | KeyForge$_C$ $\sigma$ | DKIM RSA2048 $\sigma$ | TimeForge $\sigma$ |
|---|---|---|---|---|---|
| $30 \times 65 = 1950$ | 98 | $30 \times (64 + 32) = 2880$ | $64 \times 2 + 32 = 160$ | 256 | 841 |

Table 4.1: Bandwidth costs (in bytes) of KeyForge$_B$, KeyForge$_C$, and DKIM with RSA. $\sigma$ denotes a signature.

| | Sign(ms) | Sign/s | Verify(ms) | Verify/s |
|---|---|---|---|---|
| TimeForge | 24.58 | 49.68 | 23.24 | 43 |
| KeyForge$_B$ | 0.34 | 2,932 | 3.36 | 298 |
| KeyForge$_C$ | 0.13 | 17,197 | 0.13 | 7,541 |
| RSA2048 | 0.93 | 1,075 | 0.05 | 19,966 |
| Ed25519 | 0.03 | 27,001 | 0.10 | 9,781 |

Table 4.2: Time required for a single operation in milliseconds, and the equivalent number of operations per second. KeyForge times are for a 4-level tree, RSA is from OpenSSL benchmarks.

fails, the filter alerts PostFix and the message is dropped.

**Keyserver.** The keyserver performs signing and verification, communicates with the mail filter over RPC, and publishes expired keys (for KeyForge) via a simple webserver.

### 4.5.1 Evaluation

We evaluate messaging bandwidth, expiry data bandwidth, and speed. Our primary focus is on comparison with RSA-2048: it is the signature scheme commonly used in DKIM, and so a natural benchmark for practicality in the current email ecosystem. Although more bandwidth-efficient algorithms were approved for DKIM use some months ago, (e.g., Ed25519 with a 64 B signature [173]), these schemes appear to have had limited deployment to date.[25] Nonetheless, for completeness, this section also considers Ed25519 performance.

**Bandwidth.** Table 4.1 shows bandwidth costs for various configurations of KeyForge and TimeForge. Both KeyForge implementations have a bandwidth per email that is 42% *smaller* than a DKIM RSA-2048 signature.

[25] E.g., as of October 2019, Gmail and Exchange use only RSA-2048.

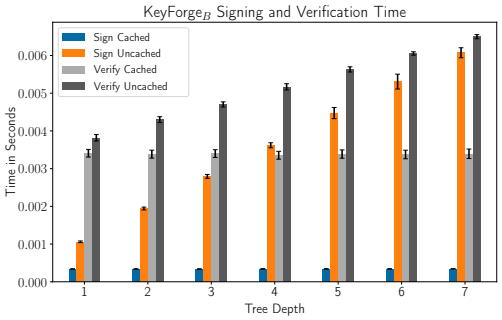



Figure 4-5: KeyForge$_B$ timings

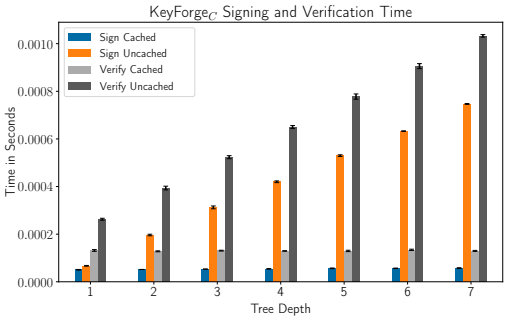



Figure 4-6: KeyForge$_C$ timings

**Speed.** To capture the range of KeyForge's possible performance, we considered two cases: (1) where the public key path is verified from scratch (e.g., in setting up a new server, or verifying messages from a new domain) and (2) where path parameters are pre-verified and cached. Figures 4-5 and 4-6 show the results. Signing is largely unaffected by tree depth when caching.

Table 4.2 provides efficiency microbenchmarks for KeyForge, TimeForge, and Ed25519 and DKIM's RSA-2048 via the OpenSSL suite's benchmark. All KeyForge benchmarks are for a 4-level tree with caching. Note that experiments were run on a laptop with power lower than a common server, so our timings may be seen as upper bounds. Performance scales linearly with the number of cores; our measurements are for a single core.

**Optimizing for KeyForge expiration bandwidth.** While the Y/M/D/Chunk configuration is easy to intuit, an equal branching factor across tree levels yields a large gain in succinctness. For example, the average size of expiry info of trees with an equal branching factor for a 2-year period is 4.5MB, 4KB, or 1.8KB for depths 1, 4, and 7.

**Discussion and analysis.** We find that KeyForge, especially KeyForge$_C$, is likely practical when using DKIM's RSA-2048 as a benchmark. In both implementations, KeyForge's signing time is *better* than RSA: KeyForge$_B$ and KeyForge$_C$ sign 2.7 and 16 times faster than RSA, respectively. KeyForge further beats RSA on signature

bandwidth per email, at just 63% or less of RSA signature size in the worst case. RSA outperforms KeyForge only on verification time: KeyForge$_C$ is still eminently practical, with verification a factor of two slower than RSA, whereas KeyForge$_B$ is an order of magnitude slower.

Verification time is unlikely to affect KeyForge's viability, as other factors such as hashing, I/O, and network latency are likely to dominate. Any hash-and-sign scheme must read the message into memory and perform a hash, so to provide a ballpark measurement of I/O and hashing, we timed OpenSSL's SHA256 on the Podesta corpus [321], stored on-disk. The average time required was 10.2ms (2.689ms std),[26] indicating that hashing and I/O is surprisingly impactful. Network latency is significant as well — SMTP requires that a sending MTA perform a *minimum* of four round trips per email.[27] A highly optimistic round-trip time of 5ms would yield of 20ms per email, not including time to send message content.

The choice between KeyForge$_B$ and KeyForge$_C$ is likely implementation dependent: while KeyForge$_B$ requires less bandwidth, its drawbacks are speed and use of non-IETF-standardized curves (unlike KeyForge$_C$).

**A note on adoption.** With an ecosystem as unwieldy as email, a reasonable concern might be that any large-scale update would be difficult. That said, now is an opportune time to propose such changes: the IETF has recently approved a new standard that will encourage MTAs to begin updating their DKIM signing and verification algorithms [173]. Further, if the community were to endorse a new standard, one could imagine large email providers (e.g., Google) displaying favorable security indicators akin to Gmail's TLS indicators [148]. Such tactics have been successful in the context of HTTPS, and we have consulted members of the IETF, W3C, and the Gmail Security team, and optimized and evaluated our prototypes with their performance priorities and concerns in mind.

[26]Email size is often pushed up by HTML formatting, embedded media, and attachments. Average email size in our corpus is 98 KB (691 KB std).

[27]SMTP messages require a round trip per command, and each email requires a MAIL, RCPT, and two DATA commands.

## 4.6 Conclusion & Discussion

In this Chapter, we presented KeyForge and TimeForge, two instantiations of signatures that become unverifiable after a fixed wall-clock time. We provided formal proofs of our construction, and motivated their use in the context of email's highly complicated and asynchronous environment. Finally, we demonstrated that the systems are promising for practical use in email — with KeyForge in particular leveraging known primitives, similar timings, and resulting in a smaller signatures than the currently used RSA2048.

In a broader sense, this work illustrated the power of taking actor's incentives as a first-order motivation in considering new cryptographic constructions. In particular, it directed us towards a problem that had not yet been solved in the literature, and the further constraints of the systems requirements forced designs that would never have been considered by cryptographers assuming the ideal case of a fresh protocol.

Remarkably, we are not alone in pursuing disincentivization (rather than prevention) as a goal in cryptography and systems security, as a number of modern developments rely on this concept to provide better security. A fascinating example is the concept of accountable algorithms [189], which focuses on the ability to cryptographically verify bad behavior and subsequently punish the offending actors. A notable success here has been Certificate Transparency, which is currently used by all major browsers to better secure the HTTPS ecosystem [214], and expansions into firmware, and users' personal keys for messaging have also seen growing adoption [212].[28] As mentioned in Chapter 2, the effectiveness of these systems will depend heavily on the ability to punish malfeasance, but providing this transparency is a fantastic first step, and further proof that disincentives — rather than pure prevention — can improve the security of larger software ecosystems.

[28] See, generally, the implementation and use-cases for Google's trillian project [295].

# Chapter 5

# Encryption and Surveillance

> "The laws of Australia prevail in Australia, I can assure you of that. The laws of mathematics are very commendable but the only law that applies in Australia is the law of Australia."
>
> — *Former Australian Prime Minister Malcolm Turnbull*

In this chapter, we explore the debate surrounding law enforcement access to encrypted data — more commonly known as the "crypto wars" — and argue that policies mandating such mechanisms would lead to insurmountable technical and policy problems. Beyond providing a technical analysis of a proposed new type of system, this work serves as a case study into how security research can help explain to policymakers that the consequences of proposed regulation may be actively harmful, and push vendor incentives away from providing user security. We demonstrate that such policy-guided security research can play a crucial role in preventing regulatory failures from causing unmitigated technological risks.

We begin in Section 5.1 with a short introduction and overview of the arguments involved in the exceptional access debate, summarizing the overall argument of the chapter. We then present background on what had been requested by law enforcement at the time and further historical background summarizing law enforcement demands as we understand them (5.2). We then discuss these effects of these demands in the context of the two most popular and rapidly growing types of platform: a messaging service and a personal electronic device such as a smartphone or tablet (5.3). To

further provide guidance to policymakers and illustrate the complexity of such a demand, Section 5.5 enumerates a set of detailed questions that policymakers must be able to answer before a demand for exceptional access is to be taken seriously.

The sections described above are adapted from the 2015 paper "Keys Under Doormats: mandating insecurity by requiring government access to all data and communications" [3], a joint work with quite a few co-authors. Changes to these sections were made to allow for cohesiveness with the rest of this dissertation, and to better explain the context in which this analysis was performed.

It is also helpful to provide an overview of how the debate has evolved since the original work was published. First, much of the context of the analysis presented here is specific to the arguments of law enforcement at the time (as explored in Section 5.2), when the debate had just entered into a new phase. These arguments have evolved significantly, as has the state of cryptography and legal environment. Second, the impact of the work speaks to the usefulness of providing such policy-oriented analyses in the first place.

To that end, there are also two new sections which have been included for the purposes of updating this work with what has happened since the original paper was released. Section 5.6 discusses the immediate reaction and impact from the original report, and reports on how the debate has since evolved. Finally, a conclusion (Section 5.7) was added to tie this work back to the rest of this dissertation. The author of this dissertation takes responsibility for all mistakes or errors in all edits and new sections.

## 5.1 Introduction & Summary

Political and law enforcement leaders in the United States, United Kingdom, India, Australia, and elsewhere have called for Internet systems to be redesigned to ensure government access to information — even encrypted information. They argue that the growing use of encryption will neutralize their investigative capabilities. They propose that data storage and communications systems must be designed for *exceptional*

*access* by law enforcement agencies. These proposals are unworkable in practice, raise enormous legal and ethical questions, and would undo progress on security at a time when Internet vulnerabilities are causing extreme economic harm.

Judging from the computer security and systems perspective, we believe that law enforcement has failed to account for the risks inherent in exceptional access systems. Based on the considerable history of failures in real-world applications, we know that such risks lurk in the technical details. In this chapter we examine whether it is technically and operationally feasible to meet law enforcement's call for exceptional access without causing large-scale security vulnerabilities. We take no issue here with law enforcement's desire to execute lawful surveillance orders when they meet the requirements of human rights and the rule of law. Our recommendation is that anyone proposing regulations should first present concrete technical requirements, which industry, academics, and the public can analyze for technical weaknesses and for hidden costs.

A similar report was written in 1997 in response to a similar but narrower and better-defined proposal called the Clipper Chip [2]. The Clipper proposal sought to have all strong encryption systems retain a copy of keys necessary to decrypt information with a trusted third party who would turn over keys to law enforcement upon proper legal authorization. The paper illustrated that, at that time, it was beyond the technical state of the art to build key escrow systems at scale. Governments kept pressing for key escrow, but Internet firms successfully resisted on the grounds of the enormous expense, the governance issues, and the risk. The Clipper Chip was eventually abandoned. A much more narrow set of law enforcement access requirements have been imposed, but only covering access to network traffic and with scope limited to regulated telecommunications companies. Still, in a small but troubling number of cases, weakness related to these requirements have emerged and been exploited by state actors and others. Those problems would have been worse had key escrow been widely deployed. And if all information applications had to have been designed and certified for exceptional access, it is doubtful that companies like Facebook and Twitter would even exist. Another important lesson from the 1990's is that the decline

in surveillance capacity predicted by law enforcement 20 years ago did not happen. Indeed, in 1992, the FBI's Advanced Telephony Unit warned that within three years Title III wiretaps would be useless: no more than 40% would be intelligible and that in the worst case all might be rendered useless [11]. Electronic surveillance by law enforcement did not come to an end. On the contrary, law enforcement has much better and more effective surveillance capabilities now than it did then.

The goal of this chapter is to similarly analyze the newly proposed requirement of exceptional access to communications in today's more complex, global information infrastructure. We find that it would pose far more grave security risks, imperil innovation, and raise thorny issues for human rights and international relations.

There are three general problems. First, providing exceptional access to communications would force a U-turn from the technical best practices now being deployed to make the Internet more secure. These practices include *forward secrecy* — where decryption keys are deleted immediately after use, so that stealing the encryption key used by a communications server would not compromise earlier or later communications. A related technique, *authenticated encryption*, uses the same temporary key to guarantee confidentiality and to verify that the message has not been forged or tampered with.

Second, building in exceptional access would substantially increase system complexity. Security researchers inside and outside government agree that complexity is the enemy of security — every new feature can interact with others to create vulnerabilities. To achieve widespread exceptional access, new technology features would have to be deployed and tested with literally hundreds of thousands of developers all around the world. This is a far more complex environment than the electronic surveillance now deployed in telecommunications and Internet access services, which tend to use similar technologies and are more likely to have the resources to manage vulnerabilities that may arise from new features. Features to permit law enforcement exceptional access across a wide range of Internet and mobile computing applications could be particularly problematic because their typical use would be surreptitious — making security testing difficult and less effective.

Third, exceptional access would create concentrated targets that could attract bad actors. Security credentials that unlock the data would have to be retained by the platform provider, law enforcement agencies, or some other trusted third party. If law enforcement's keys guaranteed access to everything, an attacker who gained access to these keys would enjoy the same privilege. Moreover, law enforcement's stated need for rapid access to data would make it impractical to store keys offline or split keys among multiple keyholders, as security engineers would normally do with extremely high-value credentials. The 2014 attacks on the United States Government Office of Personnel Management (OPM) show how much harm can arise when many organizations rely on a single institution that itself has security vulnerabilities. In the case of OPM, numerous federal agencies lost sensitive data because OPM had insecure infrastructure. If service providers implement exceptional access requirements incorrectly, the security of all of their users will be at risk.

The greatest impediment to exceptional access may be the complexities of legal jurisdiction. Building in exceptional access would be risky enough even if only one law enforcement agency in the world had it, but this is not only a US issue. The UK government passed legislation to compel communications service providers, including US-based corporations, to grant access to UK law enforcement agencies, and other countries would certainly follow suit. China has intimated that it may require exceptional access. If a British-based developer deploys a messaging application used by citizens of China, must it provide exceptional access to Chinese law enforcement?

Democracies around the world have long recognized that electronic surveillance power in the hands of government is a threat to open societies unless it is properly regulated by an effective legal system. Many — but not all — countries have enacted laws regulating the use of surveillance techniques. However, such laws alone offer little to prevent abuse; a vibrant legal system with respect for the rule of law is necessary for privacy protection in the face of ever more powerful electronic surveillance technologies. Which countries have sufficient respect for the rule of law to participate in an international exceptional access framework? How would such determinations be made? How would timely approvals be given for the millions of new products

with communications capabilities? And how would this new surveillance ecosystem be funded and supervised? The US and UK governments have fought long and hard to keep the governance of the Internet open, in the face of demands from authoritarian countries that it be brought under state control. Does not the push for exceptional access represent a breathtaking policy reversal?

The need to grapple with these legal and policy concerns could move the Internet overnight from its current open and entrepreneurial model to becoming a highly regulated industry. Tackling these questions requires more than our technical expertise as computer scientists, but they must be answered before anyone can embark on the technical design of an exceptional access system.

## 5.2 The exceptional access debate (1990's-2014)

In the 1990's, the governments of United States and a number of other industrialized countries advocated weakening encryption. Claiming that widespread encryption would be disastrous for law enforcement, the US government proposed the use of the *Clipper Chip*, an encryption device that contained a government master key to give the government access to encrypted communications. Other governments followed suit with proposals for encryption licensing that would require copies of keys to be held in escrow by *trusted third parties* — companies that would be trusted to hand over keys in response to warrants. The debate engaged industry, NGOs, academia, and others. In response, a report similar to this chapter was written on the issues raised by key escrow or trusted-third-party encryption analyzed the technical difficulties, the added risks, and the likely costs of such an escrow system [2]. That push for key escrow was abandoned in 2000 because of pressure from industry and civil society, and because of political resistance from certain US allies.

### 5.2.1 Summary of the 2014 debate

The encryption debate reopened in the 2010's with both FBI Director James Comey and UK Prime Minister David Cameron warning that encryption threatens law

enforcement capabilities, and advocating that the providers of services that use encryption be compelled by law to provide access to keys or to plaintext in response to duly authorized warrants. The goal of this work is to re-examine the impact of mandatory exceptional access in the time since the 1990's, as cryptography, systems security, and the overall Internet environment has changed significantly.[1]

The public policy debate as of 2014 was significantly hampered by the fact that law enforcement had not provided a sufficiently complete statement of their requirements for technical experts or lawmakers to analyze.[2] The following exhortation from United States FBI Director James Comey is as close as we came:

> "We aren't seeking a back-door approach. We want to use the front door, with clarity and transparency, and with clear guidance provided by law. We are completely comfortable with court orders and legal process — front doors that provide the evidence and information we need to investigate crime and prevent terrorist attacks.
>
> Cyber adversaries will exploit any vulnerability they find. But it makes more sense to address any security risks by developing intercept solutions during the design phase, rather than resorting to a patchwork solution when law enforcement comes knocking after the fact. And with sophisticated encryption, there might be no solution, leaving the government at a dead end — all in the name of privacy and network security." [177]

Prime Minister David Cameron simply wants the police to have access to everything. Speaking in the wake of the Charlie Hebdo murders in Paris, he said:

> "In our country, do we want to allow a means of communication between people which, even in extremis, with a signed warrant from the home secretary personally, that we cannot read? ... The question remains: are

[1] We follow the 1996 United States National Academy of Sciences CRISIS report in using the phrase "exceptional access" to mean that "the situation is not one that was included within the intended bounds of the original transaction." [96, p. 80]

[2] And, unfortunately, at the time of writing little has changed, as we will see in Section 5.6

> we going to allow a means of communications where it simply is not possible to do that? My answer to that question is: no, we must not." [98]

So, we must ask, is it possible to build in such exceptional access without creating unacceptable risk? In order to understand the technical and operational issues, we first review the results of the 1997 report and consider what has changed since then. We next try to clarify ideal law enforcement requirements and understand the kinds of risks that are likely to arise if these generic requirements are imposed broadly in the global Internet environment. Then, we present two technology scenarios typical of the landscape facing modern electronic surveillance. Combining what is publicly known about surveillance practices today, along with common legal requirements, we are able to present scenarios that illustrate many of the key risks that exceptional access will entail.

However, we do not suggest that our own interpretation of FBI Director Comey's stated requirements serve as a basis for regulation, but instead take these comments as a starting point for discussion. If officials in the UK or US disagree with our interpretation, we urge them to state their requirements clearly. Only then can a rigorous technical analysis be conducted in an open, transparent manner. Such analysis is crucial in a world that is so completely reliant on secure communications for every aspect of daily lives, from nations' critical infrastructure, to government, to personal privacy in daily life, to all matters of business from the trivial to the global.

### 5.2.2 Findings from the 1997 analysis of key escrow systems

We begin by reviewing the findings on the risks of key recovery/key escrow systems from the seminal paper presented almost 20 years ago by Abelson et. al. [2], which similarly examines the security risks of ensuring law enforcement access to encrypted information. The paper found that any key escrow system had basic requirements that placed substantial costs on end users, and that these costs would have been too difficult and expensive to implement. For law enforcement to have quick and reliable access to plaintext, every key escrow system required the existence of highly sensitive

yet perennially available secret keys. This requirement alone inevitably leads to an increased risk of exposure, inflated software complexity, and high economic costs.

The first downside is increased risk of a security incident. An organization that holds an escrow key could have a malicious insider that abuses its power or leaks that organization's key. Even assuming an honest agency, there is an issue of competence: cyberattacks on keyholders could easily result in catastrophic loss.

The additional complexity of a key escrow system compounds these risks. At the time, all openly proposed key escrow solutions known to the authors had major flaws that could be exploited;[3] even normal encryption was difficult to implement well, and key escrow made things much harder. Another source of complexity was the scale of a universal key recovery system — the number of agents, products, and users involved would be immense, requiring an escrow system well beyond the technology of the time. Further, key escrow threatened to increase operational complexity: a very large number of institutions would have to securely and safely negotiate targeting, authentication, validity, and information transfer for lawful information access.

All of the above factors raise costs. Risks of exposure, for instance, change the threat landscape for organizations, which must then worry about mistaken or fraudulent disclosures. The government would have increased bureaucracy to test and approve key recovery systems. Software vendors would have to bear the burden of increased engineering costs. In 1997, we found that systems enabling exceptional access to keys would be inherently less secure, more expensive, and much more complex than those without. This result helped policymakers decide against mandated exceptional access.

### 5.2.3 What changed since the 1990s?

The commercial Internet and other widely deployed global communications networks require encryption to attain even a modest level of security. An extensive debate in the 1980s and 1990s about the role of encryption came to this conclusion once before. Today, the fundamental technical importance of strong cryptography and the difficulties inherent in limiting its use to meet law enforcement purposes remain the

[3] See, e.g., [55].

same. What has changed is that the scale and scope of systems dependent on strong encryption are far greater; our society is far more reliant on far-flung digital networks that are under daily attack.

In the early 1990s, the commercialization of the Internet was being thwarted by US government controls on encryption — controls that were in many ways counterproductive to long-term commercial and national security interests. A 1996 United States National Academy of Science study concluded that, "On balance, the advantages of more widespread use of cryptography outweigh the disadvantages" [96, p. 6]. Four years later, partly in response to pressures from industry, partly in response to the loosening of cryptographic export controls by the European Union, partly because crypto export controls were declared unconstitutional by US Circuit Courts, and partly because of increasing reliance on electronic communications and commerce, the US relaxed export controls on encryption [107].

The long running policy debate over encryption actually began in the 1970s, with conflicts over whether computer companies such as IBM and Digital Equipment Corporation could export hardware and software with strong encryption, and over whether academics could publish cryptographic research freely. The debate continued through the 1980s over whether the NSA or the National Institute of Standards and Technology (NIST) would control the development of cryptographic standards for the non-national security side of the government (NIST was given the authority under the 1987 Computer Security Act). The question came to full force during the 1990s, when the US government, largely through the use of export controls, sought to prevent companies such as Microsoft and Netscape from using strong cryptography in web browsers and other software that was at the heart of the growing Internet. The end of the wars — or the apparent end — came because of the Internet boom.

In many ways, the arguments are the same as two decades ago. US government cryptographic standards — the Data Encryption Standard then, the Advanced Encryption Standard now — are widely used both domestically and abroad. We know more now about how to build strong cryptosystems, though periodically we are surprised by a break. However, the real security challenge is not the mathematics of cryptosystems;

it is engineering, specifically the design and implementation of complex software systems which include security features. Two large government efforts, healthcare.gov and the FBI Trilogy program, demonstrate the difficulties that scale and system integration pose in building large software systems. Healthcare.gov, the website implementing the president's signature healthcare program, failed badly in its initial days, unable to serve more than a tiny percentage of users [241]. A decade earlier, five years of effort spent building an electronic case file system for the FBI — an effort that cost $170 million — was abandoned as unworkable [116].

At one level, the worst has not come to pass — the power grid, the financial system, critical infrastructure in general, and many other systems all function reliably using complex software. On another level, the worst is occurring daily. Recent breaches for financial gain include: T.J. Maxx, theft of 45 million credit card records [174]; Heartland Payment Systems, compromise of 100 million credit cards [63]; Target, compromise of 40 million credit cards; Anthem, collection of names, addresses, birthdates, employment and income information, and Social Security numbers of 80 million people that could result in identity theft [4].

Attacks on government agencies are also increasing. A set of 2003 intrusions targeting US military sites collected such sensitive data as specifications for Army helicopter mission planning systems, Army and Air Force flight-planning software, and schematics for the Mars Orbiter Lander [292]. Such theft has not only been from the defense industrial base, but has included the pharmaceuticals, Internet, biotechnology and energy industries. In 2010, then Deputy Secretary of Defense William Lynn concluded, "Although the threat to intellectual property is less dramatic than the threat to critical national infrastructure, it may be the most significant cyberthreat that the United States will face over the long term" [322].

The December 2014 North Korean cyberattacks against Sony, the first publicly-visible attempt at cybersecurity blackmail by a nation-state, resulted in large headlines. But the 2011 theft from RSA/EMC of the seed keys — initial keys used to generate other keys — in hardware tokens used to provide two-factor authentication [31], and the recent theft of personnel records from the US Office of Personnel Management

are far more serious issues. The former undermined the technical infrastructure for secure systems, while the latter, by providing outsiders with personal information of government users, creates leverage for many years to come for potential insider attacks, undermining the social infrastructure needed to support secure governmental systems — including any future system for exceptional access. And while attacks against critical infrastructure have not been significant, the potential to do so has been demonstrated in test cases [178] and in an actual attack on German steel mill that caused significant damage to a blast furnace [329].

As exceptional access puts the security of Internet infrastructure at risk, the effects will be felt every bit as much by government agencies as by the private sector. Because of cost and Silicon Valley's speed of innovation, beginning in the mid-1990s, the US government moved to a commercial off the shelf (COTS) strategy for information technology equipment, including communications devices. In 2002, Information Assurance Technical Director Richard George told a Black Hat audience that "NSA has a COTS strategy, which is: when COTS products exist with the needed capabilities, we will encourage their use whenever and wherever appropriate . . ." [142]. Such a COTS solution makes sense, of course, only if the private sector technologies the government uses are secure.

Communications technologies designed to comply with government requirements for backdoors for legal access have turned out to be insecure. For ten months in 2004 and 2005, 100 senior members of the Greek government (including the Prime Minister, the head of the Ministry of National Defense and the head of the Ministry of Justice) were wiretapped by unknown parties through lawful access built into a telephone switch owned by Vodafone Greece [249]. In 2010 an IBM researcher observed that a Cisco architecture for enabling lawful interception in IP networks was insecure.[4] This architecture had been public for several years, and insecure versions had been implemented by several carriers in Europe [293]. And when the NSA examined telephone switches built to comply with government-mandated access

[4]It is worth noting that the router's design was based on standards put forth by the European Telecommunications Standards Institute.

for wiretapping, it discovered security problems with *all* the switches submitted for testing [256]. Embedding exceptional access requirements into communications technology will ensure even more such problems, putting not only private-sector systems, but government ones, at risk.

Speaking on the topic of law enforcement access and systems security, Vice Chairman of the Joint Chiefs of Staff Admiral James A. Winnefeld remarked, "But I think we would all win if our networks are more secure. And I think I would rather live on the side of secure networks and a harder problem for Mike [NSA Director Mike Rogers] on the intelligence side than very vulnerable networks and an easy problem for Mike and part of that, it's not only is the right thing to do, but part of that goes to the fact that we are more vulnerable than any other country in the world, on our dependence on cyber. I'm also very confident that Mike has some very clever people working for him, who might actually still be able to get some good work done."

While the debate over mandated law enforcement access is not new, it does take on added urgency in today's world. Given our growing dependence on the Internet, and the urgent need to make this and other digital infrastructures more secure, any move in the direction of decreased security should be looked upon with skepticism. Once before, when considering this issue, governments around the world came to the conclusion that designing in exceptional access provisions to vital systems would increase security risk and thwart innovation. As the remainder of this chapter will show, such measures are even riskier today.

## 5.3 Scenarios

Law enforcement authorities have stated a very broad requirement for exceptional access. Yet there are many details lacking including the range of systems to which such requirements would apply, the extraterritorial application, whether anonymous communications would be allowed, and many other variables. To analyze the range of security risks that may arise in commonly used applications and services, we examine two popular scenarios: encrypted real-time messaging services and devices such as

smartphones that use strong encryption to lock access to the device.

### 5.3.1 Scenario 1: Providing exceptional access to globally distributed, encrypted messaging applications

Imagine a massively distributed global messaging application on the Internet currently using end-to-end encryption. Many examples of such systems actually exist, including Signal, which is available on iPhone and Android, Off-the-Record (OTR), a cryptography-enabling plug-in for many popular computer chat programs, and the often cited TextSecure and WhatsApp. Could one provide a secure application while meeting law enforcement exceptional access requirements?

To provide law enforcement access to encrypted data, one natural approach would be to provide law enforcement direct access to keys that can be used to decrypt the data, and there is a frequently suggested and seemingly quite attractive mechanism for escrowing decryption keys. Data is typically encrypted — either for storage or transmission — with a symmetric key,[5] and many data transmission protocols (e.g., the Transport Layer Security (TLS) protocol) can operate in a mode where the data to be sent is encrypted with a symmetric key that is in turn encrypted with a public key[6] associated with the intended recipient. This encrypted symmetric key then travels with the encrypted data, and the recipient accesses the data by first using its private key to decrypt the symmetric key and then using the symmetric key to decrypt the data.

A common suggestion is to augment this approach by encrypting the symmetric key a second time — this time with a special escrowing public key. If the data is then transmitted, two encryptions of the symmetric key accompany the data — one with the public key of the intended recipient and one with a public key associated with an escrow agent. If the data has been encrypted with a symmetric key for storage rather than transmission, the symmetric key might be encrypted with the public key

[5] A symmetric key is one that is used for both encryption and decryption.

[6] A public key is used to encrypt data that can then be decrypted only by an entity in possession of an associated private key.

of an escrow agent and this escrowed key could remain with the encrypted data. If a law enforcement entity obtains this encrypted data – either during transmission or from storage – the escrow agent could be enlisted to decrypt the symmetric key, which could then be used to decrypt the data.

There are, however, three principal impediments to using this approach for third-party escrow. Two are technical and the third is procedural.

The first technical obstacle is that although the mode of encrypting a symmetric key with a public key is in common use, companies are aggressively moving away from it because of a significant practical vulnerability: *if an entity's private key is ever breached, all data ever secured with this public key is immediately compromised.* Because it is unwise to assume a network will never be breached, a single failure should never compromise all data that was ever encrypted.

Thus, companies are moving towards *forward secrecy*, an approach that greatly reduces the exposure of an entity that has been compromised. With forward secrecy, a new key is negotiated with each transaction, and long-term keys are used only for authentication. These transaction (or *session*) keys are discarded after each transaction — leaving much less for an attacker to work with. When a system with forward secrecy is used, an attacker who breaches a network and gains access to keys can only decrypt data from the time of the breach until the breach is discovered and rectified; historic data remains safe. In addition, since session keys are destroyed immediately after the completion of each transaction, an attacker must interject itself into the process of each transaction in real time to obtain the keys and compromise the data.[7]

The security benefits make clear why companies are rapidly switching to systems that provide forward secrecy.[8] However, the requirement of key escrow creates a long-term vulnerability: if *any* of the private escrowing keys are *ever* compromised, then *all* data that *ever* made use of the compromised key is permanently compromised. That is, in order to accommodate the need for surreptitious, third-party access by law enforcement agencies, messages will have to be left open to attack by anyone who can

[7]Lack of forward secrecy was identified in the 1997 paper [2] as a weakness of key escrow systems then. Since that time, the need for forward secrecy has grown substantially.

[8]See [230, 195, 156, 272, 6, 184, 237, 216, 215, 64, 200].

obtain a copy of one of the many copies of the law enforcement keys. *Thus all known methods of achieving third-party escrow are incompatible with forward secrecy.*

Innovations providing better forward secrecy also support a broad social trend: users are moving en masse to more ephemeral communications. Reasons for moving to ephemeral communications range from practical decisions by corporations to protect proprietary information from industrial espionage to individuals seeking to protect their ability to communicate anonymously and avoid attack by repressive governments. Many corporations delete email after 90 days, while individuals are moving from email to chat and using services like Snapchat where messages vanish after reading. Leading companies such as Twitter, Microsoft, and Facebook are supporting the move to transient messaging, and using modern security mechanisms to support it. This social and technical development is not compatible with retaining the means to provide exceptional access.

The second technical obstacle is that current best practice is often to use *authenticated encryption*, which provides *authentication* (ensuring that the entity at the other end of the communication is who you expect, and that the message has not been modified since being sent) as well as *confidentiality* (protecting the privacy of communications, including financial, medical, and other personal data). However, disclosure of the key for authenticated encryption to a third party means the message recipient is no longer provided with technical assurance of the communication's integrity; disclosure of the key allows the third party not only to *read* the encrypted traffic but also to *forge* traffic to the recipient and make it look as if it is coming from the original sender. Thus disclosing the key to a third party creates a new security vulnerability. Going back to the encryption methods of the 1990s, with separate keys for encryption and authentication, would not only double the computational effort required, but introduce many opportunities for design and implementation errors that would cause vulnerabilities.

The third principal obstacle to third-party key escrow is procedural and comes down to a simple question: who would control the escrowed keys? Within the US, one could postulate that the FBI or some other designated federal entity would hold

the private key necessary to obtain access to data and that judicial mechanisms would be constructed to enable its use by the plethora of federal, state, and local law enforcement entities. However, this leaves unanswered the question of what happens outside a nation's borders. Would German and French public- and private-sector organizations be willing to use systems that gave the US government access to their data — especially when they could instead use locally built systems that do not? What about Russia? Would encrypted data transmitted between the US and China need to have keys escrowed by both governments? Could a single escrow agent be found that would be acceptable to both governments? If so, would access be granted to just one of the two governments or would both need to agree to a request?

These difficult questions must be answered before any system of exceptional access can be implemented. Such an architecture would require global agreements on how escrow would be structured, often against the best interests of certain countries' domestic goals, together with mandates in virtually all nations to only sell and use compliant systems.

### 5.3.2 Scenario 2: Exceptional access to plaintext on encrypted devices such as smartphones

Imagine a smartphone platform vendor that seeks to accommodate law enforcement exceptional demands. When law enforcement comes into possession of a device, perhaps at a crime scene, and then obtains the necessary legal authorization (in the US this would be a full probably-cause warrant, as opposed to an easier-to-obtain subpoena, as a result of *Riley v. California*[9]), the agent collects a unique identifying number from the device through some service mechanism, and then sends a request to the platform vendor to unlock the device remotely or provide the keys necessary for law enforcement to unlock the device locally.

At first glance, providing access to plaintext on devices — laptop hard drives, smartphones, tablets — is straightforward. Indeed, many corporations already escrow

[9] 134 S. Ct. 2473 (2014)

device encryption keys. However, and as is frequently the case, scaling up a corporate mechanism to a global one is hard.

When encrypting device storage, the user-entered passphrase is generally not used directly as an encryption key. There are many reasons for this; from a usability perspective, the most important one is to make it easier for the user to change the passphrase. If the key were used directly, it would be a time-consuming process to decrypt and re-encrypt the entire device when the passphrase is changed. Instead, a random key is used for bulk encryption; the user-supplied key (called the Key-Encrypting Key, or KEK) is used to encrypt the random key.

To protect against brute-force attacks against the user's passphrase, the device vendor may go a step further and combine it with a device-specific unique identifier to produce the KEK. In the iPhone, the KEK is stored in a special tamper-resistant processor that limits the guess rate to once every 80 milliseconds. This protects device owners against, for example, sophisticated thieves who might try to gain access to things like banking passwords. But regardless of how the KEK is generated, obtaining access to the plaintext requires that the device-encrypting key be encrypted under some additional key or keys. These could be manufacturer-owned keys or keys belonging to one or more law enforcement agencies. Either choice is problematic [28].

If a vendor-supplied key is used, some sort of network protocol to decrypt the device key is necessary. This request must be authenticated, but how? How can the vendor have secure credentials for all of the thousands of law enforcement agencies around the world? How can the result be strongly bound to the device, to prevent unscrupulous agencies from requesting keys to devices not in their lawful possession? These are not easy requirements to meet, especially for devices that will not even boot without a valid key. They are likely to require changes to security hardware or to the software that drives it; both are difficult to do properly. Fixing glitches — especially security glitches — in deployed hardware is expensive and often infeasible.

Providing devices with law enforcement keys is equally difficult. Again, how can the vendor know who supplied the keys? How are these keys to be changed?[10] How many

[10]We note that some pieces of malware, such as Stuxnet and Duqu 2, have relied on code-signing

keys can be installed without causing unacceptable slowdowns? Another alternative is to require that law enforcement ship devices back to the vendor for exceptional access decryption. However, it will still be necessary to store over long periods of time keys that can decrypt all of the sensitive data on devices. This only shifts the risks of protecting these keys to the device manufacturers.

Some would argue that per-country keys could be a sales requirement. That is, all devices sold within the US would be required to have, say, a preinstalled FBI-supplied key. That, however, does not suffice for devices brought in by travelers — and those are the devices likely to be of interest in terrorism investigations. A requirement that keys be installed at the border is also problematic. There are no standard input ports or key-loading mechanisms; furthermore, it would expose American travelers to malware installed by border guards in other countries [242, 45].

### 5.3.3 Summary of risks from the two scenarios

Designing exceptional access into today's digitally-connected consumer electronics, web-based information services and mobile applications will give rise to a range of critical security risks. First, major efforts that the industry is making to improve security will be undermined and reversed. Providing access over any period of time to thousands of law enforcement agencies will necessarily increase the risk that intruders will hijack the exceptional access mechanisms. If law enforcement needs to look backwards at encrypted data for one year, then one year's worth of data will be put at risk. If law enforcement wants to assure itself real time access to communications streams, then intruders will have an easier time getting access in real time, too. This is a trade-off space in which law enforcement cannot be guaranteed access without creating serious risk that criminal intruders will gain the same access.

Second, the challenge of guaranteeing access to multiple law enforcement agencies in multiple countries is enormously complex. It is likely to be prohibitively expensive and also an intractable foreign affairs problem.

Simple requirements can yield simple solutions (e.g. a door lock). But the

keys issued to legitimate companies. When a key is compromised, it must be replaced.

requirements of law enforcement access to encrypted data are inherently complex and, as we have already shown, nearly contradictory. Complex or nearly contradictory requirements yield brittle, often-insecure solutions. As NSA's former head of research testified in 2013:

> "When it comes to security, complexity is not your friend. Indeed it has been said that complexity is the enemy of security. This is a point that has been made often about cybersecurity in a variety of contexts including, technology, coding and policy. The basic idea is simple: as software systems grow more complex, they will contain more flaws and these flaws will be exploited by cyber adversaries." [133]

We have a very real illustration of the problem of complexity in a recent analysis of one of the most important security systems on the Internet: SSL/TLS. Transport Layer Security (TLS) and its predecessor Secure Socket Layer (SSL) are the mechanisms by which the majority of the web encrypts its traffic — every time a user logs into a bank account, makes an electronic purchase, or communicates over a social network, that user is trusting SSL/TLS to function properly. All a user needs to know of all of this complexity is that the lock or key icon shows up in the browser window. This indicates that the communication between the user and the remote website is secure from interception.

Unfortunately, writing code that correctly implements such cryptographic protocols has proven difficult; weakened protections makes it harder still. For instance, OpenSSL, the software used by about two-thirds of websites to do TLS encryption, has been plagued with systems-level bugs resulting in catastrophic vulnerabilities. The now-infamous Heartbleed bug was caused by a missing bounds check, an elementary programming error that lurked in the code for two years, leaving 17% of *all* websites vulnerable to data theft. More recent vulnerabilities, however, were caused by legacy restrictions on the exportation of cryptographic algorithms, dating back to the Crypto Wars. The fact that there are so many different implementations of TLS, all of which have to interoperate to make the Web secure, has proven to be a serious source of

security risk [53]. Website operators are reluctant to switch to more secure protocols if this will lose them even a few percent of prospective customers who are still using old software, so vulnerabilities introduced deliberately during the Crypto Wars have persisted to this day. Introducing complex new exceptional access requirements will similarly add more security bugs that will lurk in our software infrastructure for decades to come.

Third, there are broader risks for poorly deployed surveillance technology. Exceptional access mechanisms designed for law enforcement use have been exploited by hostile actors in the past. Between 1996 and 2006, it appears that insiders at Telecom Italia enabled the wiretapping of 6,000 people, including business, financial, and political leaders, judges, and journalists [247]. In a country of 60 million, this means that no major business or political deal could be considered truly private. The motivation here appeared to be money, including the possibility of blackmail. As we mentioned earlier, from 2004 to 2005, the cell phones of 100 senior members of the Greek government, including the Prime Minister, the head of the Ministry of National Defense, the head of the Ministry of Justice, and others were subject to ongoing and unauthorized surveillance. Vodafone Greece had purchased a telephone switch from Ericsson. The Greek phone company had not purchased wiretapping capabilities, but these were added during a switch upgrade in 2003. Because Vodafone Greece had not arranged for interception capabilities, the company did not have the ability to access related features, such as auditing. Nevertheless, someone acting without legal authorization was able to activate the intercept features and keep them running for ten months without being detected. The surveillance was uncovered only when some text messages went awry. Although the techniques of how it was done are understood, individuals who were behind the surveillance remain unknown [249].

Finally, there are the broader costs to the economy. Economic growth comes largely from innovation in science, technology, and business processes. At present, technological progress is largely about embedding intelligence — software and communications — everywhere. Products and services that used to be standalone now come with a mobile phone app, an online web service, and business models that involve either ads

or a subscription. Increasingly these are also "social", so you can chat to your friends and draw them into the vendor's marketing web. Countries that require these new apps and web services to have their user-to-user communications functions authorized by the government will be at a significant disadvantage. The market advantage from an open Internet environment gives real benefits not just economically but in terms of soft power and moral leadership. The open Internet has long been a foreign policy goal of the US and its allies for good reason. The West's credibility on this issue was damaged by the Snowden revelations, but can recover.

## 5.4 Security impact of common law enforcement requirements with exceptional access

As there is no specific statement of law enforcement requirements for exceptional access, we consider what we understand to be a very general set of electronic surveillance needs applicable in multiple jurisdictions around the world. Our goal here is to understand the general nature of security risks associated with the application of exceptional access requirements in the context of traditional categories of electronic surveillance. Law enforcement agencies in different countries have presented different requirements at different times, which we will treat under four headings: access to communications content, access to communications data, access to content at rest, and covert endpoint access. All types of access must be controlled and capable of being audited according to local legal requirements; for example, under the requirements of US law, one must respect the security and privacy of non-targeted communications.[11]

### 5.4.1 Access to communications content

Most police forces are permitted to access suspect data. In countries with respect for the rule of law, such access is carefully regulated by statute and supervised by an independent judiciary, though most of the world's population do not enjoy such legal

[11] In the USA, 47 USC 1002(a)(4)

protections. Law enforcement access might be to a central database of unencrypted messages where this exists at a central provider. Where there is no central database, such as for a telephone or video call, the police must tap the communication in real time. How might an exceptional access requirement be implemented to enable for access to communications content? If the data is encrypted, the most obvious mechanism to allow for police access would require that traffic between Alice in country X and Bob in country Y would have its session key also encrypted under the public keys of the police forces in both X and Y, or of third parties trusted by them. This, however, raises serious issues.

First, any escrow requirement will restrict other important security functionality such as forward secrecy, the use of transient identities, and strong location privacy. As illustrated in the scenario analysis above, an exceptional access requirement overlaid on the traditional content surveillance will put the security of the content at risk. To the extent that capabilities exist to provide law enforcement exceptional access, they can be abused by others.

Second, the global nature of the Internet makes compliance with exceptional access rules both hard to define and hard to enforce. If software sold in country X will copy all keys to that country's government, criminals might simply buy their software from countries that don't cooperate; thus, US crooks might buy their software from Russia. And if software automatically chooses which governments to copy using a technique such as IP geolocation, how does one prevent attacks based on location spoofing? While it is possible to design mobile phone systems so that the host jurisdictions have access to the traffic (so long as the users do not resort to VoIP), this is a much harder task for general-purpose messaging applications.

Third, one might have to detect or deter firms that do not provide exceptional access, leading to issues around certification and enforcement. For example, if the US or the UK were to forbid the use of messaging apps that are not certified under a new escrow law, will such apps be blocked at the national firewall? Will Tor then be blocked, as in China? Or will it simply become a crime to use such software? And what is the effect on innovation if every new communications product must go through

government-supervised evaluation against some new key escrow protection profile?

### 5.4.2 Access to communications transactional data

Communications transactional data traditionally meant call detail records and, since mobile phones became common, caller location history; it was obtained by subpoena from phone companies, and is used in the investigation of serious violent crimes such as murder, rape, and robbery. Communications transactional data (now popularly known as 'metadata') remains widely available as service providers keep it for some time for internal purposes. However, police forces outside the US complain that the move to globalized messaging services makes a lot of data harder to obtain because of questions about whether services operating in a third country are subject to local legal requirements. A new UK surveillance law may require message service firms like Apple, Google, and Microsoft to honor such requests expeditiously and directly as a condition of doing business in the UK. So will there be uniform provisions for access to communications data subject to provisions for warrants or subpoenas, transparency, and jurisdiction?

As already noted, determining location is not trivial, and cheating (using foreign software, VPNs, and other proxies) could be easy. Criminals would turn to noncompliant messaging apps, raising issues of enforcement; aggressive enforcement might impose real costs on innovation and on industry generally.

### 5.4.3 Access to data at rest

Communications data are one instance of the general problem of access to data at rest. Almost all countries allow their police forces access to data. Where basic rule of law is in place, access is under the authority of a legal instrument such as a warrant or subpoena, subject to certain limits. Many corporations already insist on escrowing keys used to protect corporate data at rest (such as BitLocker on corporate laptops). So this is one field with an already deployed escrow "solution": a fraud investigator wanting access to a London rogue trader's laptop can simply get a law enforcement

officer to serve a decryption notice on the bank's CEO. But still, many of the same problems arise. Suspects may use encryption software that does not have escrow capability, or may fail to escrow the key properly, or may claim they have forgotten the password, or may actually have forgotten it. The escrow authority may be in another jurisdiction, or may be a counterparty in litigation. In other words, what works tolerably well for corporate purposes or in a reasonably well-regulated industry in a single jurisdiction simply does not scale to a global ecosystem of highly diverse technologies, services, and legal systems.

Another complex case of access to data at rest arises when the data is only present on, or accessible via, a suspect's personal laptop, tablet, or mobile phone. At present, police officers who want to catch a suspect using Tor services may have to arrest him or her while his/her laptop is open and a session is live. Law enforcement agencies in some countries can get a warrant to install malware on a suspect's computer. Such agencies would prefer antivirus companies not to detect their malware; some might even want the vendors to help them, perhaps via a warrant to install an upgrade with a remote monitoring tool on a device with a specific serial number. The same issues arise with this kind of exceptional access, along with the issues familiar from covert police access to a suspect's home to conduct a surreptitious search or plant a listening device. Such exceptional access would gravely undermine trust and would be resisted vigorously by vendors.

## 5.5 Principles at stake and unanswered questions

With people's lives and liberties increasingly online, the question of whether policy makers should support law enforcement demands for guaranteed access to private information has a special urgency, and must be evaluated with clarity. From a public policy perspective, there is an argument for giving law enforcement the best possible tools to investigate crime, subject to due process and the rule of law. But a careful scientific analysis of the likely impact of such demands must distinguish what might be desirable from what is technically possible. In this regard, a proposal to

regulate encryption and guarantee law enforcement access centrally is analogous to a requirement that all airplanes can be controlled from the ground. While this might be desirable in the case of a hijacking or a suicidal pilot, a clear-eyed assessment of how one could design such a capability reveals enormous technical and operational complexity, international scope, large costs, and massive risks — so much so that such proposals, though occasionally made, are not really taken seriously.

We have shown that current law enforcement demands for exceptional access would likely entail very substantial security risks, engineering costs, and collateral damage. If policy-makers believe it is still necessary to consider exceptional access mandates, there are technical, operational, and legal questions that must be answered in detail before legislation is drafted. From our analysis of the two scenarios and general law enforcement access requirements presented earlier, we offer this set of questions.

### 5.5.1 Scope, limitations, and freedoms

The first set of questions that an exceptional access proposal must address concerns the scope of applicability of the exceptional access requirement, any limitations on the mandate, and what user freedoms would remain protected under such proposals. Questions such as these arise in this category:

1. Are all systems that use encryption covered, or just some? Which ones?

2. Do all online communications and information platforms have to provide access to plain text, or merely provide keys to agencies that had already collected ciphertext using technical means?

3. Would individuals, corporations, nonprofit institutions, or governments be allowed to deploy additional encryption services on top of those systems with exceptional access? Would those user-installed systems also have to meet exceptional access requirements?

4. Would machine-to-machine systems be covered? What about large-scale sensor networks (a.k.a. Internet of Things) and industrial control (SCADA) systems?

Much information exchange is from one machine to another, such as communicating personal health data from a sensor to a smartphone, field-based agricultural sensing devices to tractors, or load balancing controls in electric power, gas, oil and water distribution systems.

5. How would cross-border regulatory differences be resolved? Would technology developers have to meet different exceptional access requirements in each jurisdiction where their systems are used? Or would there be a globally harmonized set of regulatory requirements?

6. How can the technical design of an exceptional access system prevent mass surveillance that would covertly violate the human rights of entire populations, while still allowing covert targeted surveillance of small numbers of suspects?

7. Would there be an exception for research and teaching?

8. Could companies refuse to comply with exceptional access rules based on a fear of violating human rights?

9. Would anonymous communications, widely recognized as vital to democratic societies, be allowed?

### 5.5.2 Planning and design

Designing the technology and planning the administrative procedures that would be needed to implement a comprehensive exceptional access system raises many questions:

1. What are the target cost and benefit estimates for such a program? No system is cost-free and this one could be very expensive, especially if it has to accommodate a large number of providers, such as today's millions of independent app developers.

2. What security and reliability measures would be established for the design? How would system prototypes be tested? How long would companies have to comply with exceptional access rules?

3. How would existing services and products be treated if they do not comply with exceptional access rules? Would providers have to redesign their systems? What if those systems cannot accommodate exceptional access requirements?

4. Who would be involved in the design of the systems and procedures — just the US and UK government, or would other governments be invited to participate? Could foreign technology providers such as Huawei participate in the design discussions?

5. Would the technical details of the program be made public and open for technical review? What level of assurance would be provided for the design?

6. We note that it generally takes many years after a cryptographic protocol is published before it is deemed secure enough for actual use. For example, the Needham-Schroeder public-key protocol, first published in 1978 [226], was discovered to have security flaw only in 1995 by Gavin Lowe (17 years later!) [203].

### 5.5.3 Deployment and operation

Once regulations are established and technical design parameters set, there would remain questions about how systems would be deployed, who would supervise and regulate compliance, and how the design of the system would evolve to address inevitable technical and operational bugs that emerge. We know of no system that is designed perfectly the first time, and it is well understood that maintenance, support, and evolution of existing systems constitutes a major expense.

1. Who would supervise compliance? Would an existing regulatory agency such as the FCC be given jurisdiction over the entire process? How would other countries regulate US domestic and foreign services? Would there be a global harmonization of rules regulation and enforcement? Would the International Telecommunications Union have a role in setting and enforcing requirements?

2. Would global technical standards be required? How would these be developed and enforced? How would be such standards be changed/improved/patched? Would

traditional standards bodies such as the UN International Telecommunications Union T-sector or ISO set standards, or would the world look to Internet standards bodies such as the IETF and the World Wide Web Consortium? How would the world converge on one set of standards?

3. Would the US government provide reference software libraries implementing the desired functionality? Would other governments be invited to collaborate in designing and testing those libraries? Or might other governments develop their own software for implementation by services provided in those countries?

4. Would programs and apps need to be certified before they were allowed to be sold? Who would test or certify that programs produced operate as intended?

5. Who would be liable if the plaintext-disclosure mechanisms were buggy (either in design or in implementation), causing the disclosure of all citizens' information? More generally, what would happen when (not if) critical secret information was revealed, such as the private keys that allow encrypted data to be read by anyone, that destroyed the privileged position of law enforcement?

6. How many companies would withdraw from markets where exceptional access was mandated in ways that clashed with their business strategies or the rights of users in other countries, as Google already has done from China and Russia?

### 5.5.4 Evaluation, assessment, and evolution

Large systems exist because successful systems evolve and grow. Typically, this evolution happens through interaction guided by the institution (software company, government agency, or open-source community) responsible for the system. If all systems that communicate must in future evolve subject to an exceptional access constraint, there will be real costs which are hard to quantify, since the question of who exactly would be responsible for establishing and policing the exceptional access constraint is not clear. Systems such as medical systems that need to maintain a safety case or flight control systems that need to maintain not just a safety case but

also need to meet real-time performance requirements, evolve less quickly and at more cost. However these costs questions are answered, the following further issues will arise.

1. What oversight program would be required to monitor the effectiveness, cost, benefits, and abuse of exceptional access?

2. What sunset provisions would be build into legislation for such a program? What conditions would be in place for its termination (e.g., for lack of sufficient benefit, for excessive cost, or for excessive abuse)?

3. One unintended consequence of such a program may be a much-reduced use of crypto altogether. This would further weaken our already fragile and insecure information infrastructure, so how do we incentivize companies to continue encrypting sensitive user communications?

4. A further unintended consequence of such a program might be to make the US and other participating countries less welcoming to technological innovation; diminishing or displacing innovation may have consequences for economic growth and national security. How will these economic impacts be assessed before an exceptional access program is mandated? Further, what economic effect would be considered too impactful for exceptional access to be considered worthwhile?

## 5.6 Response to the original work & evolution of the debate

A preprint of the original paper that presented our analysis in the sections above was published on July 7, 2015, and covered in the New York Times [243]. The work appears to have had a marked impact on the debate, including the tactics of the actors involved. Here we discuss how the debate has since evolved, and where it will likely go in the future.

The paper heavily influenced the legislative debate in congress. Following the paper's release, the Senate Judiciary committee held a hearing on "Going Dark: Encryption, Technology, and the Balance Between Public Safety and Privacy," in which the work was quoted by both those giving testimony [288] and by the minority chairperson [240]. Various authors of the work later provided testimony to both the House Judiciary [193] and House Energy and Commerce [162] committees further explaining the problems with exceptional access. A joint, bipartisan working group between these two committees released a report largely adopting the analysis presented in this chapter [146].

Beyond congress, the debate had been brought to the attention of the public, civil society, and industry. An open letter stating that the U.S. should "Reject any law, policy, or mandate that would undermine our security" was drafted and signed by various civil society organizations like the American Civil Liberties Union, Center for Democracy and Technology, the Electronic Frontier Foundation, and Human Rights Watch, as well as by tech companies including Twitter, Dropbox, and Cloudflare [270]. Academic institutions including the IEEE and ACM released similar statements of support [5, 86]. Speaking in a personal capacity, Ashkan Soltani, then the FTC's chief technologist, stated his personal support for strong encryption [32]. The debate became entered into the public consciousness so significantly that political candidates in both the Democratic and Republican 2016 presidential primaries had to answer questions about the subject [312]

By October, the Obama Administration made it clear that it would not support a mandate. FBI Director Comey released a statement confirming that "The administration has decided not to seek a legislative remedy now, but it makes sense to continue the conversations with industry," and similar statements from the White House seemed to corroborate this new tact [118]. Though the administration's announcement appeared to concede the status quo, this peace was largely temporary.

A month following the administration's decision, the New York District Attorney's office releases a report explicitly narrowing law enforcement's request to data at rest on mobile devices [227]. In February 2016, the FBI convinces a judge to order

Apple to create a system to unlock an iPhone 5C used by one of the assailants in a mass shooting [250]. Apple flatly refuses, and signals their intent to fight the order through public statements and by hiring a lawyer that had previously argued before the Supreme Court [252]. The case was abruptly withdrawn after the FBI announced it had gotten access to the device via exploitation of a previously unknown vulnerability [49].

The use of software exploitation tools to provide such access is called *lawful hacking* [43]. In the time since the Apple order, lawful hacking appears to have been more widely adopted, with notable uses being discovered for internet services [180] as well as data at rest [179].

While lawful hacking and a legislative stalemate became the norm in the United States, the debate abroad moved in quite a different direction. Laws were passed in both Australia [289] and the United Kingdom [172] that allow government and law enforcement to issue Technical Capability Notices (TCNs) to companies, requiring them to modify their systems or build some interception tool for law enforcement. Such TCNs are kept secret, and, at the time of writing, it is unknown what TCNs have been issued, how they have caused companies to modify or restrict the security properties of their systems.

### 5.6.1 Technical proposals

Over the years since the original work was published, researchers have attempted to provide prototype systems with various technical and societal trade-offs. At the time of writing, these are research prototypes of system designs, not true implementations — none of these are operable today and do not appear to solve many of the systems, policy, or jurisdictional problems covered in previous sections.

There have been a few formal proposals to provide access to device data at rest, though with serious drawbacks. Varia et. al.'s system [326] allows for access to a device, but only after expending an extreme amount of compute via a hashcash like crypto puzzle, so as to make large-scale surveillance more difficult. Other systems [269, 235] assume significant hardware support that allow a device to provide plaintext, but

results in the system becoming inoperable after the fact. Neither of these solve many of the jurisdictional or legal questions proposed above, nor do they solve the technical issues introduced by the added complexity of the systems [150, 42].

There have been few new proposals for end to end encrypted messaging, and, at the time of writing, no notable proposals for encrypted client/server communication. A notable straw man comes from two GCHQ operatives, who suggest that companies can simply add an extra hidden key to encrypted group messaging platforms like iMessage or Signal [199]. This idea, dubbed the "Ghost Key" or "Ghost User" protocol, will not work on many end to end encrypted systems as they provide out of band methods of verifying the keys used (as in Whatsapp or Signal) [151].

A recent line of work [109, 154] considers the problem of *franking* in encrypted messaging systems. These protocols break certain deniability guarantees of the cryptography to allow for better moderation: they allow attribution of messages even after an arbitrary period, but only with the cooperation of a centralized service provider. Facebook has deployed one such system, based on a secret HMAC key [122]. The idea is to provide accountability in the case of harassment or abuse in end to end encrypted communication, while still providing deniability otherwise with the assumption that the service provider itself is not malicious.

At the time of writing, another technique called *client side scanning* has become the subject of intense debate in the security community. Client side scanning is a general term for systems that leverage the user's device to scan messages or photos locally, and then secretly flags the results to a third party or the device software vendor [191]. At the time of writing, Apple has announced that it intends to deploy a client side scanning tool for photos uploaded to its iCloud system explicitly to handle the problem of Child Sexual Abuse Material (CSAM). The announcement has already met with significant opposition from technical experts [152] and civil society [210]. It is unclear how client side scanning solves many of the problems described previous sections, or if it fits into the same framework. future work must carefully consider the tradeoffs of such a system, and cover this particular issue in more depth than is possible here.

## 5.7 Conclusion

Since the release of the Keys Under Doormats paper, Law enforcement has repeatedly called for exceptional access mandates, both in congress and through press conferences. A remarkable aspect of all of this is the lack of evidence for the need for such provisions, and a general failure to provide further guidance for law enforcement need. Indeed, it is unclear if law enforcement truly understands their own needs — at the time of writing, the most non-anecdotal analysis provided by law enforcement on the problems they have with encrypted data is limited to the current number of locked phones in their possession, which they quoted as 7,800. What, exactly, these devices were for, how they were related to a case, if the material was truly paramount, which devices these were, or if some other supplementary information or methodology for access would have been helpful, has never been explained. Worse, it turned out that the FBI had vastly over-inflated the number of devices it had trouble with; in reality, the FBI was forced to admit they had only 1,200.

There is a direct connection between the observation of software security as a credence quality and the challenges of implementing an exceptional access system. In this chapter, we showed that though users will have to rely on systems with exceptional access built in, there will be very little in the way of incentives for developers to get security properties right, and may lead developers to simply shy away from implementing certain kinds of cryptography at all. Indeed, almost immediately after it became publicly known that the FBI filed its suit against Apple, Amazon removed full disk encryption from its popular e-reader devices [131].

Worse, the many impediments associated with the current opacity of UK and Australian laws, as well as the refusal of any government to state clear technical or even operational requirements for such systems, could leave implementers grasping at straws to fulfill unintelligible demands. The UK's Investigatory Powers Act, for instance, does not sufficiently limit the kinds of requests that come down to any useful technical specificity, and the results of a change are often subtle — regulators are unlikely to know the technical pitfalls that a particular request might have.

More generally, given that users will often have to trust the government to help protect them from security faults *caused* by industry, this kind of regulation may hurt user confidence regardless of the actual order. How can users trust their devices to act in their best interest if the system's design was altered by opaque surveillance laws imposed by the exact same regulator that is supposed to be protecting them from that developer's (potentially) bad design?[12]

While the previous case studies focus on directing security research to identify current flaws or create new systems, this final chapter demonstrates how leveraging an understanding of technical security measures, as well as legal and societal dimensions of an ecosystem, can help *prevent* impending regulation that would invariably misalign vendor incentives and introduce future vulnerabilities. Indeed, the original paper has been cited in both the fairness and accountability literature [189], and in Rogaway's treatise on the moral character of cryptographic work [262] for exactly this reason. Given the fact that this debate continues to evolve, and that new research and development must continue, there is a dire need for future analysis along similar lines. As computer scientists and security experts, it is our responsibility to remain engaged in the dialogue with all parts of our governments, to help discern the best path through these complex questions.

[12]Of course, law enforcement and consumer protection agencies may be organizationally separate, though it is unclear how a user might know this.

# Chapter 6

# Conclusion

> "Governments of the Industrial World, you weary giants of flesh and steel, I come from Cyberspace, the new home of Mind. On behalf of the future, I ask you of the past to leave us alone. You are not welcome among us. You have no sovereignty where we gather."
>
> — *John Perry Barlow*

Throughout this dissertation, we have examined the concept of security research as a mechanism for realigning incentives of market actors, considering a number of different perspectives and styles of security analysis. In this final chapter, we take an inventory of these case studies and briefly present a discussion of how we might begin to further support this style of research moving forward.

When we fail to accept the larger economic or legal context, we often miss opportunities to impactfully improve security. Our analysis of voting systems (Chapter 3) demonstrated that indicia of market failure like nonexistent standards, poor labeling, and limited transparency can be helpful in both selecting targets for technical analysis, and in explaining why these incidents keep occurring. Challenging the view that security should be considered only as a tool for provably halting attacks, the cryptographic constructions presented in Chapter 4 were instead motivated explicitly by the need to disincentivize bad behavior, a tactic that is of growing importance in the cryptographic literature. Finally, our security analysis of law enforcement proposals for exceptional access to encrypted data (Chapter 5) demonstrates the need for technical security research to assist decision makers in government when the need arises.

At the heart of each chapter is an argument that the larger sociotechnical context mattered as much as the technology involved, and that myopia resisting this interdisciplinarity actively stalls the field's ability to make progress in these crucial areas. If we accept this premise, then it is important to consider what, exactly, academics and policymakers can do to improve the situation. We offer some thoughts in this direction below:

For policymakers, we must begin to assume that external scrutiny is paramount to security and privacy, and favor it above checklist-style regulations and standards. It is also crucial that regulations be improved so as to remove impediments that security researchers often face in perform these analyses. And, for any proposed technical mandate on security-critical systems, inviting third-party academic analysis can help in ameliorating future unintentional failures.

Unfortunately, the history of security research is littered with companies leveraging bad policy to prevent useful third party analysis. In the U.S., two laws called the Digital Millennium Copyright Act (DMCA) and the Computer Fraud Abuse Act (CFAA) present ambiguous language restricting this kind of analysis, and both have caused a chilling effect toward security research. So much so that, despite the obvious public benefit of the research in the voting Chapter, the authors were forced to maintain legal counsel [92]. A similar challenge for researchers is attaining *access* to relevant data. At the time this dissertation is being written, Facebook has come under fire for denying researchers access to independently inspect their ad ecosystem, resulting in statements of protest from civil society, academics, and the FTC [198].

For academics in computer science, a key problem is incentivizing this kind of public good focused security research. As highlighted in this dissertation's introduction, academic reviewing in security does not always account for societal impact, and, as a result, often does not incentivize this sort of work. Indeed, though the paper was eventually accepted, an actual review of one of the studies presented in this dissertation's voting systems chapter (which had already materially affected policy in a number of states) included the following: "In reading the paper, I don't see the contents as research, novel, or science." Though this was but one out of five reviews —

the other four being overwhelmingly positive and, more importantly, the paper was ultimately accepted — the quote does still highlight the need for better education and inclusion of this sort of policy and impact-driven analysis. To this reviewer, we hope this dissertation serves as sufficient rebuttal.

It is worth noting that *both* policymakers, and academics have further work to do here. Chapter 5 began with an epigraph quoting Australian prime minister Malcolm Turnbull's assertion that the laws of mathematics do not supersede the laws of Australia, a (frankly bizarre) claim he made in response to a reporter's question about exceptional access mandates. This chapter begins with a quote from John Perry Barlow's "Declaration of Independence of Cyberspace," one of the founders of the Electronic Frontier Foundation (EFF).[1]

Taken for the values the two statements represent, rather than their literal interpretation, neither are completely correct, nor are they completely wrong. Countries can very obviously regulate industries that operate in their borders, regardless of their use of the Internet, and regulating *any* aspect of computing is arguably regulating what math can be done, if only through the system's defaults. Remarkably, Barlow has since updated this position with further nuance [36], and the EFF has openly called for increased government regulation in the case of fighting the sale of malicious software and the adoption of digital consumer privacy and protection laws [139, 255].

If we are to improve security, and have influence on these larger, sociotechnical debates moving forward, we must begin to include motivation, methodologies, and contexts from other disciplines into our work, rather than just focusing on the technology itself. We play a key role in this larger ecosystem, as third party measurement, auditing, and mechanism design are crucial to the improvement of real world systems. As computer scientists and security experts, it is our responsibility to remain engaged in the dialogue, to help discern the best path through these complex questions — Our security, privacy, and the future of democracy relies on our ability to do so.

[1] Though Barlow himself was not a computer scientist, this document serves as a fairly good proxy for an approach often espoused by technologists.

# Appendix A

# Further Text on Deniability

## A.1 Comparison Table

| | Messaging | | | Email | |
|---|---|---|---|---|---|
| | **OTR v3** | **Signal** | **MLS** | **DKIM** | **KeyForge & TimeForge** |
| Attributable? | non-attributable | | | attributable | non-attributable |
| Interactive? | yes (see Remark 3) | | | no | |
| Known endpoints? | yes | | | no | |
| Synchronous? | yes | no | yes | no | |
| Routing | centralized | centralized | decentralized | fully decentralized w/ unknown endpoints | |
| Federated? | no | no | yes | yes | |
| Authentication bandwidth | 464B* | 64B† | 64B [233] | 256B (RSA2048) | KeyForge: 146B<br>TimeForge: 1096B |
| Bandwidth Per User | N/A‡ | $O(n^2)$§ | $O(n^2)$ | O(1) | |

Table A.1: Comparison of Messaging Schemes

* This is the size of the initial Diffie-Hellman handshake (which would amortize over multiple messages), not including any per-message costs.

† The exact signature scheme is implementation-dependent. This data reflects both the current Signal implementation and the best bandwidth among implementations of which we are aware, which use Ed25519-based signatures. [234]

‡ OTR does not support group chat.

§ WhatsApp has an optimization that reduces message traffic; published documentation leaves unclear if this affects authentication handshake bandwidth. [316]

Table A.1 lays out characteristics of KeyForge, DKIM-based email and other types of non-attributable messaging schemes. As also explained in Section 4.1, non-attributable email presents a very different set of constraints from other types of non-email messaging schemes, so direct performance comparisons between these

settings are not meaningful. Table A.1 highlights in more detail how the email setting differs from other types of messaging.

*Remark* 3. All other non-attributable messaging schemes of which we are aware require some sort of interactivity — either by having the sender know the endpoint a priori or by having the receiver communicate back to the sender before a message is sent. While Signal's updates to OTR allow it to be more asynchronous than some other protocols, Signal is still interactive in that it requires communication of key material to known endpoints. Such interactivity makes these messaging schemes inappropriate for use in email.

## A.2 ARC

Authenticated Received Chain (ARC) is an experimental and (at the time of writing) largely unimplemented IETF standard that aims to address the issues caused by indirect email flows. Legitimate modification of messages in transit may occur in a number of circumstances. For example, an MTA that is also a virus scanner may remove malicious attachments, and mailing lists may prepend a name to the subject of an email. Unfortunately, any alteration of the body or headers invalidates the original signature. ARC acts as an attestation by an intermediary a subset of DKIM, SPF, DMARC, or previous ARC signatures were verifiable before content was modified. To do so, the intermediary adds its own signature to the header of the message, along with metadata about what was verified.

**Complications from ARC.** Arc would pose minor complications for non-attributability if widely adopted, as third-party MTAs sign emails in transit. Though here presented as a modification of DKIM, KeyForge and TimeForge can be easily exgended to accommodate ARC; the third-party signer uses an FFS for signing, publishes expiry information, and offers a forge-on-request protocol.

## A.3 Preventing real-time attacks requires interaction

**Claim 1.** *In the store-and-forward model (see §4.1.2), where final-destination addresses and keys may be unknown to senders, any email protocol that (1) proves (with soundness) the sending domain's identity to recipients and (2) is secure against real-time attacks must involve* interaction*, i.e., recipient communication to the sending server after receipt of an initial message.*[1]

*Proof (sketch).* By assumption, there is a verification procedure $\mathsf{V}$ that the receiver $R$ may run on the single-message protocol transcipt, and $\mathsf{V}$ will (with overwhelming probability) output 1 if the claimed sending domain is correct and 0 otherwise. Since the final destination may be unknown to the sender, $\mathsf{V}$'s behavior cannot depend on any information associated specifically with the receiver, such as key material. That is, $\mathsf{V}$'s inputs must consist only of the protocol transcript and public information. It follows that $R$ may share the received message(s) with any third party $T$, who then becomes able to run $\mathsf{V}$ for itself on the same inputs as $R$ would use. By soundness, $\mathsf{V}$'s output on these inputs must suffice to convince $T$ of the sending domain's identity *unless* $R$ has the ability to non-interactively forge protocol transcripts that appear to be from this sending domain, in real time — say, using an algorithm $\mathsf{F}$. But because the final destination may be unknown to the sender, the behavior of any such $\mathsf{F}$ algorithm cannot depend on any information associated specifically with the receiver, so the $\mathsf{F}$ algorithm could also be run by other parties to successfully forge protocol transcripts that appear to be from this sending domain. This contradicts soundness (Condition (1) from the theorem statement). □

Among other things, this rules out immediate recipient forgeability by timed-authentication approaches like TESLA [244, 245] (also mentioned in §4.3 under "Discussion of alternative approaches").

[1] In fact, a version of Claim 1 extends to any *unidirectional* protocol, which may involve multiple messages but all sent by the same party, as long as there is realistic nontrivial variance in network delays. For simplicity, the proof sketch is for single-message protocols.

# Bibliography


[1] AAAS Center for Scientific Evidence in Public Issues. *Letter to Governors and Secretaries of State on the insecurity of online voting*. https://www.aaas.org/programs/epi-center/internet-voting-letter. Apr. 2020 (cit. on p. 73).

[2] Hal Abelson, Ross N. Anderson, Steven Michael Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, et al. "The risks of key recovery, key escrow, and trusted third-party encryption". In: (1997). URL: http://academiccommons.columbia.edu/catalog/ac:127127 (visited on 07/05/2015) (cit. on pp. 165, 168, 170, 177).

[3] Harold Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, and Peter G. Neumann. "Keys Under Doormats: Mandating Insecurity by Requiring Government Access to All Data and Communications". In: *Journal of Cybersecurity* 1.1 (2015), pp. 69–79 (cit. on pp. 14, 164).

[4] Reed Abelson and Matthew Goldstein. "Anthem Hacking Points to Security Vulnerability of Health Care Industry". In: *The New York Times* (Feb. 2015). ISSN: 0362-4331. URL: http://www.nytimes.com/2015/02/06/business/experts-suspect-lax-security-left-anthem-vulnerable-to-hackers.html (visited on 07/05/2015) (cit. on p. 173).

[5] ACM US Public Policy Council. *Policy Statement on Mandatory Engineered Law Enforcement Access to Information Infrastructure and Devices*. Apr. 12, 2018. URL: https://www.acm.org/binaries/content/assets/public-policy/usacm/2018-usacm-statement-law-enforcement-access.pdf (cit. on p. 193).

[6] Adam Langley. *Protecting data for the long term with forward secrecy*. Nov. 2011. URL: http://googleonlinesecurity.blogspot.com/2011/11/protecting-data-for-long-term-with.html (visited on 07/06/2015) (cit. on p. 177).

[7] Tony Adams, Andrew W. Appel, Arlene Ash, Steven M. Bellovin, Matt Blaze, Duncan Buell, Michael D. Byrne, Jack Cable, Jeremy Clark, Sandy Clark, Stephen Checkoway, Richard DeMillo, David L. Dill, Zakir Durumeric, Aleksander Essex, David Evans, Ariel J. Feldman, Edward W. Felten, Bryan Ford, Joshua M. Franklin, Juan E. Gilbert, J. Alex Halderman, Joseph Lorenzo Hall, Harri Hursti, Neil Jenkins, David Jefferson, Douglas W. Jones, Philip Kortum, Carl E. Landwehr, Maggie MacAlpine, Bruce McConnell, Patrick McDaniel, Walter Mebane, Eric Mill, David Mussington, Peter G. Neumann, Lyell Read,

Ronald L. Rivest, Aviel D. Rubin, Bruce Schneier, Alexander A. Schwarzmann, Hovav Shacham, Micah Sherr, Barbara Simons, Kevin Skoglund, Michael A. Specter, Alex Stamos, Philip B. Stark, Jacob Stauffer, Camille Stewart, Rachel Tobac, Giovanni Vigna, Poorvi L. Vora, Dan S. Wallach, Tarah Wheeler, Eric Wustrow, Ka-Ping Yee, and Daniel M. Zimmerman. *Scientists say no credible evidence of computer fraud in the 2020 election outcome, but policymakers must work with experts to improve confidence*. Letter. [https://perma.cc/5DSM-7ZJ8]. Nov. 2020. URL: https://www.mattblaze.org/papers/election2020.pdf (cit. on pp. 16, 38).

[8] Ben Adida. "Helios: Web-based Open-Audit Voting." In: *USENIX security symposium*. Vol. 17. 2008, pp. 335–348 (cit. on pp. 46, 71).

[9] Ben Adida, David Chau, Susan Hohenberger, and Ronald L. Rivest. "Lightweight email signatures". In: *International Conference on Security and Cryptography for Networks*. Springer, 2006, pp. 288–302 (cit. on pp. 109, 153).

[10] Ben Adida, Susan Hohenberger, and Ronald L. Rivest. "Lightweight Encryption for Email". In: *Steps to Reducing Unwanted Traffic on the Internet Workshop, SRUTI'05*. USENIX Association, 2005. URL: https://www.usenix.org/conference/sruti-05/lightweight-encryption-email (cit. on pp. 118, 150).

[11] Advanced Telephony Unit, Federal Bureau of Investigation. *Telecommunications Overview, slide on "Encryption Equipment"*. 1992. URL: https://www.cs.columbia.edu/~smb/Telecommunications_Overview_1992.pdf (cit. on p. 166).

[12] Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. "reCAPTCHA: Human-based character recognition via web security measures". In: *Science* 321.5895 (2008), pp. 1465–1468 (cit. on p. 80).

[13] George A. Akerlof. "The Market for "Lemons": Quality Uncertainty and the Market Mechanism". In: *Uncertainty in Economics*. Elsevier, 1978, pp. 235–251 (cit. on p. 25).

[14] Devdatta Akhawe, Frederik Braun, Francois Marier, and Joel Weinberger. *Subresource Integrity*. en. https://www.w3.org/TR/SRI/. 2016. (Visited on 05/21/2020) (cit. on p. 104).

[15] Amazon Web Services. *S3 Object Lock overview*. https://docs.aws.amazon.com/AmazonS3/latest/dev/object-lock-overview.html. (Visited on 05/12/2020) (cit. on pp. 74, 83).

[16] Amazon Web Services. *Share an Object with Others*. https://docs.aws.amazon.com/AmazonS3/latest/dev/ShareObjectPreSignedURL.html (cit. on p. 82).

[17] Amazon Web Services. *What Is AWS CloudTrail?* https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-user-guide.html (cit. on p. 74).

[18] Ross Anderson. "Why Cryptosystems Fail". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. 1993, pp. 215–227 (cit. on p. 13).

[19] Ross Anderson. "Why Information Security Is Hard-an Economic Perspective". In: *Seventeenth Annual Computer Security Applications Conference*. IEEE, 2001, pp. 358–365 (cit. on pp. 14, 23).

[20] Ross Anderson and Tyler Moore. "Information Security Economics–and Beyond". In: *Annual International Cryptology Conference*. Springer, 2007, pp. 68–91 (cit. on pp. 14, 23, 26).

[21] Andrew Yang. *Modernize Voting*. en-US. https://www.yang2020.com/policies/modernize-voting/. (Visited on 01/07/2020) (cit. on p. 41).

[22] Android. *Android keystore system*. en. https://developer.android.com/training/articles/keystore. (Visited on 01/21/2020) (cit. on p. 54).

[23] Android. *Settings.Secure | Android Developers*. en. https://developer.android.com/reference/android/provider/Settings.Secure. (Visited on 01/18/2020) (cit. on p. 54).

[24] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, and Michalis Kallitsis. "Understanding the Mirai Botnet". In: *26th ${$USENIX$}$ Security Symposium (${$USENIX$}$ Security 17)*. 2017, pp. 1093–1110 (cit. on p. 30).

[25] *Apktool*. ibotpeaches.github.io/Apktool. (Visited on 01/13/2020) (cit. on p. 59).

[26] Andrew W. Appel, Richard A. DeMillo, and Philip B. Stark. "Ballot-Marking Devices (BMDs) Cannot Assure the Will of the Voters". In: *Election Law Journal* 19.3 (2020) (cit. on pp. 45, 88).

[27] Andrew Appel and Philip B. Stark. "Evidence-Based Elections: Create a Meaningful Paper Trail, Then Audit". In: *Georgetown Law Technology Review* 4 (July 2020), pp. 523–541 (cit. on pp. 42, 45).

[28] Apple. *iOS Security on iOS 8.3 or Later*. Tech. rep. Apr. 2015. URL: https://www.apple.com/business/docs/iOS%5C_Security%5C_Guide.pdf (cit. on p. 180).

[29] D. F. Aranha and C. P. L. Gouvêa. *RELIC is an Efficient LIbrary for Cryptography*. https://github.com/relic-toolkit/relic (cit. on p. 156).

[30] Diego F. Aranha, Laura Fuentes-Castaneda, Edward Knapp, Alfred Menezes, and Francisco Rodriguez-Henriquez. "Implementing pairings at the 192-bit security level". In: *International Conference on Pairing-Based Cryptography*. Springer, 2012, pp. 177–195 (cit. on p. 156).

[31] Arthur Coviello. *Open Letter from Arthur Coviello, Executive Chairman, RSA, Security Division of EMC, to RSA customers*. Mar. 2011 (cit. on p. 173).

[32] Ashkan Soltani. *The Virtues of Strong Enduser Device Controls*. Federal Trade Commission. Aug. 26, 2015. URL: https://www.ftc.gov/news-events/blogs/techftc/2015/08/virtues-strong-enduser-device-controls (visited on 08/17/2021) (cit. on p. 193).

[33] Associated Press. *DKIM verification script*. https://github.com/associatedpress/verify-dkim (cit. on p. 109).

[34] Kenneth A. Bamberger and Deirdre K. Mulligan. "Privacy on the Books and on the Ground". In: *Stanford Law Review* 63 (2010), p. 247 (cit. on p. 33).

[35] Jonathan Band and Masanobu Katoh. *Interfaces on Trial 2.0*. The MIT Press, 2011 (cit. on p. 22).

[36] John Perry Barlow. *John Perry Barlow: Is Cyberspace Still Anti-Sovereign?* Cal Alumni Association. Feb. 12, 2018. URL: https://alumni.berkeley.edu/california-magazine/march-april-2006-can-we-know-everything/jp-barlow-cyberspace-still-anti (visited on 08/27/2021) (cit. on p. 200).

[37] Kevin Beaumont. *Somebody sent me a link to another Github account, with the author name listed at Voatz. It has hardcoded username and passwords.* https://twitter.com/GossiTheDog/status/1026904510386585600. Aug. 2018. (Visited on 01/10/2020) (cit. on p. 49).

[38] Kevin Beaumont. *The Voatz website is running on a box with out of date SSH, Apache (multiple CVSS 9+), PHP etc.* https://twitter.com/GossiTheDog/status/1026607447996354561. Aug. 2018. (Visited on 01/10/2020) (cit. on p. 49).

[39] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. "Compact E-Cash and Simulatable VRFs Revisited". In: *Pairing-Based Cryptography '09*. 2009 (cit. on p. 150).

[40] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. "P-signatures and Noninteractive Anonymous Credentials". In: *TCC 2008*. 2008. URL: http://dx.doi.org/10.1007/978-3-540-78524-8_20 (cit. on p. 150).

[41] Susan Bell, Josh Benaloh, Michael D Byrne, Dana DeBeauvoir, Bryce Eakin, Philip Kortum, Neal McBurnett, Olivier Pereira, Philip B Stark, Dan S Wallach, et al. "STAR-Vote: A secure, transparent, auditable, and reliable voting system". In: *2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 13)*. 2013 (cit. on pp. 46, 71).

[42] Steven M. Bellovin, Matt Blaze, Dan Boneh, Susan Landau, and R. R. Rivest. *Analysis of the CLEAR Protocol per the National Academies' Framework*. Technical report CUCS-003-18. Department of Computer Science, Columbia ..., 2018 (cit. on p. 195).

[43] Steven M. Bellovin, Matt Blaze, Sandy Clark, and Susan Landau. "Lawful Hacking: Using Existing Vulnerabilities for Wiretapping on the Internet". In: *Nw. J. Tech. & Intell. Prop.* 12 (2014), p. 1 (cit. on p. 194).

[44] Steven Michael Bellovin. "Spamming, phishing, authentication, and privacy". In: (2004) (cit. on p. 109).

[45] Ben Thompson. "UAE Blackberry update was spyware". In: *BBC* (July 2009). URL: http://news.bbc.co.uk/2/hi/8161190.stm (visited on 07/06/2015) (cit. on p. 181).

[46] Josh Benaloh. "Simple Verifiable Elections." In: *EVT* 6 (2006), pp. 5–5 (cit. on p. 49).

[47] Josh Benaloh. "Verifiable Secret-Ballot Elections". https://www.microsoft.com/en-us/research/publication/verifiable-secret-ballot-elections/. PhD thesis. Yale University, Sept. 1987 (cit. on p. 46).

[48] Ruha Benjamin. *Race After Technology*. 2019 (cit. on p. 34).

[49] Katie Benner and Eric Lichtblau. "U.S. Says It Has Unlocked iPhone Without Apple". In: *The New York Times. Technology* (Mar. 28, 2016). ISSN: 0362-4331. URL: https://www.nytimes.com/2016/03/29/technology/apple-iphone-fbi-justice-department-case.html (visited on 08/18/2021) (cit. on p. 194).

[50] Tara Siegel Bernard. "Equifax Breach Affected 147 Million, but Most Sit Out Settlement". en-US. In: *The New York Times* (Jan. 2020). ISSN: 0362-4331. URL: https://www.nytimes.com/2020/01/22/business/equifax-breach-settlement.html (visited on 06/29/2021) (cit. on p. 13).

[51] Matthew Bernhard, Josh Benaloh, J Alex Halderman, Ronald L Rivest, Peter YA Ryan, Philip B Stark, Vanessa Teague, Poorvi L Vora, and Dan S Wallach. "Public evidence from secret ballots". In: *2nd Intl. Joint Conf. on Electronic Voting*. E-Vote-ID. 2017 (cit. on pp. 43, 46, 72).

[52] Matthew Bernhard, Allison McDonald, Henry Meng, Jensen Hwa, Nakul Bajaj, Kevin Chang, and J. Alex Halderman. "Can Voters Detect Malicious Manipulation of Ballot Marking Devices?" In: *41st IEEE Symposium on Security and Privacy*. 2020 (cit. on pp. 88, 103).

[53] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. "A messy state of the union: Taming the composite state machines of TLS". In: *IEEE Symposium on Security and Privacy*. 2015. URL: https://www.smacktls.com/smack.pdf (visited on 07/05/2015) (cit. on p. 183).

[54] Trail of Bits. *Our Full Report on the Voatz Mobile Voting Platform | Trail of Bits Blog*. https://blog.trailofbits.com/2020/03/13/our-full-report-on-the-voatz-mobile-voting-platform/. (Visited on 06/03/2020) (cit. on p. 101).

[55] Matt Blaze. "Protocol failure in the escrowed encryption standard". In: *Proceedings of the 2nd ACM Conference on Computer and Communications Security*. ACM. 1994, pp. 59–67 (cit. on p. 171).

[56] Matt Blaze, Jake Braun, and Cambridge Global Advisors. "DEFCON 25 Voting Machine Hacking Village". In: *Proceedings of DEFCON, Washington DC* (2017), pp. 1–18 (cit. on p. 40).

[57] Jenny Blessing, Michael A. Specter, and Daniel J. Weitzner. *You Really Shouldn't Roll Your Own Crypto: An Empirical Study of Vulnerabilities in Cryptographic Libraries*. July 10, 2021. arXiv: 2107.04940 [cs]. URL: http://arxiv.org/abs/2107.04940 (visited on 08/27/2021) (cit. on p. 31).

[58] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. "Verifiable Delay Functions". In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*. 2018, pp. 757–788. DOI: 10.1007/978-3-319-96884-1_25. URL: https://doi.org/10.1007/978-3-319-96884-1%5C_25 (cit. on p. 146).

[59] Dan Boneh and Xavier Boyen. "Short Signatures Without Random Oracles". In: *Advances in Cryptology - EUROCRYPT 2004*. Ed. by Christian Cachin and Jan L. Camenisch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 56–73. ISBN: 978-3-540-24676-3 (cit. on p. 151).

[60] Dan Boneh, Xavier Boyen, and Hovav Shacham. "Short Group Signatures". In: *Advances in Cryptology – CRYPTO 2004*. Ed. by Matt Franklin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 41–55. ISBN: 978-3-540-28628-8 (cit. on p. 152).

[61] *Bonito Boats, Inc. v. Thunder Craft Boats, Inc.* 1988 (cit. on p. 22).

[62] Nikita Borisov, Ian Goldberg, and Eric Brewer. "Off-the-record Communication, or, Why Not to Use PGP". In: *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*. WPES '04. Washington DC, USA: ACM, 2004, pp. 77–84. ISBN: 1-58113-968-3. DOI: 10.1145/1029179.1029200. URL: http://doi.acm.org/10.1145/1029179.1029200 (cit. on pp. 17, 117, 118).

[63] Brian Krebs. *Security Fix - Payment Processor Breach May Be Largest Ever*. Jan. 2009. URL: http://voices.washingtonpost.com/securityfix/2009/01/payment%5C_processor%5C_breach%5C_may%5C_b.html (visited on 07/06/2015) (cit. on p. 173).

[64] Peter Bright. *Microsoft expands the use of encryption on Outlook, OneDrive*. July 2014. URL: http://arstechnica.com/security/2014/07/microsoft-expands-the-use-of-encryption-on-outlook-onedrive/ (visited on 07/06/2015) (cit. on p. 177).

[65] *Bryan Finney on Technology and Voting by Neither Free Nor Fair?* Anchor. Feb. 5, 2021. URL: https://anchor.fm/political-economy-forum/episodes/Bryan-Finney-on-Technology-and-Voting-eq0ir1 (visited on 08/23/2021) (cit. on p. 101).

[66] Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. "XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions". In: *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*. Ed. by Bo-Yin Yang. Vol. 7071. Lecture Notes in Computer Science. Springer, 2011, pp. 117–129. ISBN: 978-3-642-25404-8. DOI: 10.1007/978-3-642-25405-5_8. URL: https://doi.org/10.1007/978-3-642-25405-5%5C_8 (cit. on p. 139).

[67] B. Bunz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy (SP)*. Vol. 00, pp. 319–338. DOI: 10.1109/SP.2018.00020. URL: doi.ieeecomputersociety.org/10.1109/SP.2018.00020 (cit. on pp. 150, 151).

[68] Joy Buolamwini and Timnit Gebru. "Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification". In: *Proceedings of Machine Learning Research* 81 (2018), p. 1 (cit. on p. 34).

[69] Jack Burbank, David Mills, and William Kasch. *Network Time Protocol Version 4: Protocol and Algorithms Specification*. en. https://tools.ietf.org/html/rfc5905 [https://perma.cc/428T-HN3Y]. URL: https://tools.ietf.org/html/rfc5905 (visited on 05/04/2019) (cit. on p. 120).

[70] *Burson v. Freeman*. 504 U.S. 191 (U.S. Supreme Court). 1992 (cit. on p. 43).

[71] Joseph A. Calandrino, Ariel J. Feldman, J. Alex Halderman, David Wagner, Harlan Yu, and William P. Zeller. "Source code review of the Diebold voting system". In: *University of California, Berkeley under contract to the California Secretary of State* (2007) (cit. on p. 40).

[72] John Callas, Eric Allman, Miles Libbey, Michael Thomas, Mark Delany, and Jim Fenton. *DomainKeys Identified Mail (DKIM) Signatures*. en. URL: https://tools.ietf.org/html/rfc4871 (visited on 08/29/2019) (cit. on p. 108).

[73] Jon Callas. *[ietf-dkim] DKIM Key Sizes*. http://mipassoc.org/pipermail/ietf-dkim/2016q4/017207.html [https://perma.cc/K8LM-KJS7]. Oct. 2016. (Visited on 02/27/2019) (cit. on pp. 19, 109, 117).

[74] Jon Callas. *[ietf-dkim] DKIM Key Sizes*. http://mipassoc.org/pipermail/ietf-dkim/2016q4/017195.html [https://perma.cc/7NNX-QJUK]. Oct. 2016. (Visited on 02/27/2019) (cit. on pp. 19, 109).

[75] Jon Callas. *[Ietf-dkim] Thinking About DKIM and Surveillance*. https://mailarchive.ietf.org/arch/msg/ietf-dkim/eWKbWdYmkX_d2ki_lAbczVSj8qY [https://perma.cc/DQF6-SQNZ] (cit. on p. 109).

[76] Jan Camenisch, Rafik Chaabouni, and abhi shelat abhi. "Efficient Protocols for Set Membership and Range Proofs". In: *Advances in Cryptology - ASIACRYPT 2008*. Ed. by Josef Pieprzyk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 234–252. ISBN: 978-3-540-89255-7 (cit. on p. 151).

[77] Jan Camenisch and Anna Lysyanskaya. "Signature schemes and anonymous credentials from bilinear maps". In: *Advances in Cryptology–CRYPTO 2004*. 2004 (cit. on p. 150).

[78] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. "Deniable Encryption". In: *Advances in Cryptology — CRYPTO '97*. Ed. by Burton S. Kaliski. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 90–104. ISBN: 978-3-540-69528-8 (cit. on p. 129).

[79] John C. Carney. *Sixth Modification of the Declaration of a State of Emergency for the State of Delaware Due to a Public Health Threat*. https://governor.delaware.gov/wp-content/uploads/sites/24/2020/03/Sixth-Modification-to-State-of-Emergency-03242020.pdf. Mar. 2020 (cit. on p. 39).

[80] Timothy N. Cason and Lata Gangadharan. "Environmental Labeling and Incomplete Consumer Information in Laboratory Markets". In: *Journal of Environmental Economics and Management* 43.1 (2002), pp. 113–134 (cit. on p. 31).

[81] Charles Cazabon. *getmail version 5*. http://pyropus.ca/software/getmail. URL: http://pyropus.ca/software/getmail (visited on 05/03/2019) (cit. on p. 116).

[82] D. Chaum. "Secret-ballot receipts: True voter-verifiable elections". In: *IEEE Security Privacy* 2.1 (Jan. 2004), pp. 38–47. ISSN: 1558-4046. DOI: 10.1109/MSECP.2004.1264852 (cit. on p. 46).

[83] David Chaum. "Surevote: technical overview". In: *Proceedings of the workshop on trustworthy elections (WOTE'01)*. 2001 (cit. on p. 71).

[84] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L Rivest, Peter YA Ryan, Emily Shen, and Alan T Sherman. "Scantegrity II: End-to-End Verifiability for Optical Scan Election Systems using Invisible Ink Confirmation Codes." In: *EVT* 8 (2008), pp. 1–13 (cit. on pp. 46, 71).

[85] Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham. "A Systematic Analysis of the Juniper Dual EC Incident". In: *23rd ACM Conference on Computer and Communications Security*. CCS. 2016 (cit. on p. 75).

[86] Chris Brantley. *New IEEE Position Statement Supports Strong Encryption | IEEE Global Public Policy*. June 24, 2018. URL: https://globalpolicy.ieee.org/new-ieee-position-statement-supports-strong-encryption-for-confidentiality-and-data-integrity/ (visited on 08/17/2021) (cit. on p. 193).

[87] Kevin Collier. *West Virginia backtracks on using smartphone voting app in state primary*. en. Library Catalog: www.nbcnews.com. URL: https://www.nbcnews.com/tech/tech-news/west-virginia-backtracks-using-smartphone-voting-app-state-primary-n1145571 (visited on 06/03/2020) (cit. on p. 101).

[88] Nicholas Confessore. "Cambridge Analytica and Facebook: The Scandal and the Fallout So Far". In: *The New York Times. U.S.* (Apr. 4, 2018). ISSN: 0362-4331. URL: https://www.nytimes.com/2018/04/04/us/politics/cambridge-analytica-scandal-fallout.html%20[%5Curl%7Bhttps://perma.cc/47W7-A79S%7D] (visited on 06/28/2021) (cit. on p. 13).

[89] Sunny Consolvo. "Privacy and Security Practices of Individuals Coping with Intimate Partner Abuse". In: (2017) (cit. on p. 71).

[90] Don Coppersmith and Markus Jakobsson. "Almost Optimal Hash Sequence Traversal". In: *Financial Cryptography, 6th International Conference, FC 2002, Southampton, Bermuda, March 11-14, 2002, Revised Papers*. Ed. by Matt Blaze. Vol. 2357. Lecture Notes in Computer Science. Springer, 2002, pp. 102–119. ISBN: 3-540-00646-X. DOI: 10.1007/3-540-36504-4_8. URL: https://doi.org/10.1007/3-540-36504-4%5C_8 (cit. on p. 139).

[91] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. "A Large-Scale Analysis of the Security of Embedded Firmwares". In: *23rd USENIXSecurity Symposium (USENIX Security 14)*. 2014, pp. 95–110 (cit. on pp. 13, 27).

[92] Andrew Crocker and Gilens Naomi. *Nathan Van Buren V. United States, Brief of Amici Curiae: Computer Security Researchers, Electronic Frontier Foundation, Center for Democracy & Technology, Bugcrowd, Rapid7, Scythe, and Tenable in Support of Petitioner*. July 8, 2020. URL: https://www.eff.org/files/2020/07/08/19-783_eff_security_researchers_amici_brief_.pdf (visited on 06/19/2021) (cit. on pp. 19, 199).

[93] D. Crocker. *Internet Mail Architecture*. en. https://tools.ietf.org/html/rfc5598. 2009. URL: https://tools.ietf.org/html/rfc5598 (visited on 02/15/2019) (cit. on p. 114).

[94] Chris Culnane, Mark Eldridge, Aleksander Essex, and Vanessa Teague. "Trust Implications of DDoS Protection in Online Elections". In: *2nd Intl. Joint Conf. on Electronic Voting*. E-Vote-ID. 2017 (cit. on p. 91).

[95] Cyber Independent Testing Lab. *Binary Hardening in IoT Products*. https://cyber-itl.org/2019/08/26/iot-data-writeup.html. Aug. 26, 2019. (Visited on 06/18/2021) (cit. on p. 13).

[96] Kenneth W. Dam, Herbert S. Lin, et al. *Cryptography's role in securing the information society*. National Academies Press, 1996. ISBN: 0-309-05475-3. (Visited on 07/06/2015) (cit. on pp. 169, 172).

[97] Michael R. Darby and Edi Karni. "Free Competition and the Optimal Amount of Fraud". In: *The Journal of Law and Economics* 16.1 (Apr. 1973), pp. 67–88. ISSN: 0022-2186, 1537-5285. DOI: 10.1086/466756. URL: https://www.journals.uchicago.edu/doi/10.1086/466756 (visited on 02/22/2021) (cit. on pp. 26, 33, 35).

[98] David Cameron. *PM: spy agencies need more powers to protect Britain*. Jan. 2015. URL: https://embed.theguardian.com/embed/video/uk-news/video/2015/jan/12/david-cameron-spy-agencies-britain-video (cit. on p. 170).

[99] David Jefferson, Duncan Buell, Kevin Skoglund, Joe Kiniry, and Joshua Greenbaum. *What We Don't Know About the Voatz "Blockchain" Internet Voting System*. https://cse.sc.edu/~buell/blockchain-papers/documents/WhatWeDontKnowAbouttheVoatz_Blockchain_.pdf. May 2019 (cit. on pp. 49, 106).

[100] S. Delaune, S. Kremer, and M. Ryan. "Coercion-resistance and receipt-freeness in electronic voting". In: *19th IEEE Computer Security Foundations Workshop (CSFW'06)*. ISSN: 2377-5459. July 2006, 12 pp.–42. DOI: 10.1109/CSFW.2006.8 (cit. on p. 43).

[101] Democracy Live. *Disclosure Policy*. Accessed Oct. 10, 2020. https://democracylive.com/disclosure-policy/. (Visited on 10/11/2020) (cit. on pp. 96, 100).

[102] Democracy Live. *OmniBallot Frequently Asked Questions*. https://sites.omniballot.us/kcd/app/faq. (Visited on 05/12/2020) (cit. on pp. 39, 75).

[103] Democracy Live. *OmniBallot Online is an online sample ballot and electronic ballot system*. https://democracylive.com/omniballot-online/. (Visited on 05/12/2020) (cit. on pp. 74, 75).

[104] Democracy Live. *Post Election Security Audit: King Conservation District*. Feb. 2020 (cit. on p. 74).

[105] Democracy Live. *Privacy policy*. https://democracylive.com/privacy-policy/. June 2020 (cit. on p. 100).

[106] Mario Di Raimondo and Rosario Gennaro. "New Approaches for Deniable Authentication". In: *Journal of Cryptology* 22.4 (Oct. 2009), pp. 572–615. ISSN: 1432-1378. DOI: 10.1007/s00145-009-9044-3. URL: https://doi.org/10.1007/s00145-009-9044-3 (cit. on p. 129).

[107] W. Diffie and S. Landau. "The Export of Cryptography in the 20th Century and the 21st". In: *The History of Information Security: A Comprehensive Handbook*. Ed. by Karl De Leeuw and Jan Bergstra. The Netherlands: Elsevier, 2007, pp. 725–736 (cit. on p. 172).

[108] Disclose.io. *Election CVD Terms*. https://github.com/disclose/terms/blob/master/vertical/core-terms-US-2020-ELECTIONS.md. June 2020 (cit. on pp. 96, 100).

[109] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. "Fast Message Franking: From Invisible Salamanders to Encryptment". In: *CRYPTO (1)*. Vol. 10991. Lecture Notes in Computer Science. Springer, 2018, pp. 155–186 (cit. on p. 195).

[110] Emily Dreyfuss. *Is Your Wobbly, Illegible Touchscreen Signature Still You?* Wired. https://www.wired.com/story/is-your-wobbly-illegible-touchscreen-signature-still-you/. May 2019 (cit. on p. 79).

[111] Uwe Dulleck and Rudolf Kerschbamer. "On Doctors, Mechanics, and Computer Specialists: The Economics of Credence Goods". In: *Journal of Economic Literature* 44.1 (Mar. 2006), pp. 5–42. ISSN: 0022-0515. DOI: 10.1257/002205106776162717. URL: https://www.aeaweb.org/articles?id=10.1257/002205106776162717 (visited on 04/27/2020) (cit. on p. 30).

[112] Uwe Dulleck, Rudolf Kerschbamer, and Matthias Sutter. "The Economics of Credence Goods: An Experiment on the Role of Liability, Verifiability, Reputation, and Competition". In: *American Economic Review* 101.2 (2011), pp. 526–55 (cit. on p. 30).

[113] Cynthia Dwork, Moni Naor, and Amit Sahai. "Concurrent Zero-knowledge". In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. STOC '98. Dallas, Texas, USA: ACM, 1998, pp. 409–418. ISBN: 0-89791-962-9. DOI: 10.1145/276698.276853. URL: http://doi.acm.org/10.1145/276698.276853 (cit. on p. 129).

[114] Susan Dzieduszycka-Suinat, Judy Murray, Joseph R. Kiniry, Daniel M. Zimmerman, Daniel Wagner, Philip Robinson, Adam Foltzer, and Shpatar Morina. *The Future of Voting: End-to-End Verifiable Internet Voting*. U.S. Vote Foundation. https://www.usvotefoundation.org/E2E-VIV. 2015 (cit. on p. 46).

[115] Peter Eckersley. "How unique is your web browser?" In: *10th Privacy Enhancing Technologies Symposium*. PETS. 2010 (cit. on p. 92).

[116] Dan Eggen and Griff Witte. "The FBI's Upgrade That Wasn't". en-US. In: *The Washington Post* (Aug. 2006). ISSN: 0190-8286. URL: http://www.washingtonpost.com/wp-dyn/content/article/2006/08/17/AR2006081701485.html (visited on 07/06/2015) (cit. on p. 173).

[117] Election Assistance Commission. *EAC Releases Annual Grant Expenditure Report*. https://www.eac.gov/news/2017/08/16/eac-releases-annual-grant-expenditure-report. Aug. 2017. (Visited on 01/12/2020) (cit. on p. 40).

[118] Ellen Nakashima and Andrea Peterson. *Obama Administration Opts Not to Force Firms to Decrypt Data — for Now - The Washington Post*. Oct. 8, 2015. URL: https://www.washingtonpost.com/world/national-security/obama-administration-opts-not-to-force-firms-to-decrypt-data--for-now/2015/10/08/1d6a6012-6dca-11e5-aa5b-f78a98956699_story.html (visited on 08/17/2021) (cit. on p. 193).

[119] Jeffrey C. Ely and Juuso Välimäki. "Bad Reputation". In: *The Quarterly Journal of Economics* 118.3 (2003), pp. 785–814 (cit. on p. 30).

[120] Winand Emons. "Credence Goods and Fraudulent Experts". In: *The RAND Journal of Economics* 28.1 (1997). Publisher: [RAND Corporation, Wiley], pp. 107–119. ISSN: 0741-6261. DOI: 10.2307/2555942. URL: https://www.jstor.org/stable/2555942 (visited on 05/21/2020) (cit. on p. 28).

[121] Ittay Eyal and Emin Gün Sirer. "Majority is not enough: Bitcoin mining is vulnerable". In: *International conference on financial cryptography and data security*. Springer, 2014, pp. 436–454 (cit. on p. 63).

[122] Facebook. *Messenger Secret Conversations Technical Whitepaper*. https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf. July 2016 (cit. on p. 195).

[123] Timothy J. Feddersen and Thomas W. Gilligan. "Saints and Markets: Activists and the Supply of Credence Goods". In: *Journal of Economics & Management Strategy* 10.1 (2001), pp. 149–171 (cit. on p. 34).

[124] Joan Feigenbaum and Daniel J. Weitzner. "On the Incommensurability of Laws and Technical Mechanisms: Or, What Cryptography Can't Do". In: *Cambridge International Workshop on Security Protocols*. Springer, 2018, pp. 266–279 (cit. on p. 15).

[125] Ariel J. Feldman, J. Alex Halderman, and Edward W. Felten. "Security Analysis of the Diebold AccuVote-TS Voting Machine". In: *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*. EVT'07. Boston, MA: USENIX Association, 2007, p. 2 (cit. on p. 40).

[126] Ed Felten. *The Linux Backdoor Attempt of 2003*. Freedom to Tinker. https://freedom-to-tinker.com/2013/10/09/the-linux-backdoor-attempt-of-2003/. 2013 (cit. on p. 75).

[127] David Fifield. "A better zip bomb". In: *13th USENIX Workshop on Offensive Technologies (WOOT 19)*. 2019 (cit. on p. 59).

[128] Hilda L. Fontana. *Authentication Failure Reporting Using the Abuse Reporting Format*. en. https://tools.ietf.org/html/rfc6591 [https://perma.cc/5MTF-ZD8P]. URL: https://tools.ietf.org/html/rfc6591 (visited on 05/04/2019) (cit. on p. 143).

[129] Forrest Centi. *The Denver Mobile Voting Pilot: A Report*. https://cyber-center.org/wp-content/uploads/2019/08/Mobile-Voting-Audit-Report-on-the-Denver-County-Pilots-FINAL.pdf. Aug. 2018. (Visited on 01/11/2020) (cit. on pp. 51, 60).

[130] Jason Franceschi-Bicchierai, Lorenzo Koebler, and Emanuel Maiberg. *A Mobile Voting App That's Already in Use Is Filled With Critical Flaws*. en. Library Catalog: www.vice.com. Mar. 2020. URL: https://www.vice.com/en_us/article/n7jk9x/mobile-voting-app-voatz-severe-security-vulnerabilities (visited on 06/03/2020) (cit. on p. 102).

[131] Lorenzo Franceschi-Bicchierai. "Amazon Quietly Removes Encryption Support from Its Gadgets". In: *Vice* (Mar. 3, 2016). URL: https://www.vice.com/en/article/mg7vmn/amazon-removes-device-encryption-fire-os-kindle-phones-and-tablets (visited on 08/27/2021) (cit. on p. 196).

[132] Thomas Claburn Francisco. "Compsci Guru Wants 'right to Be Forgotten' for Old Email, Urges Google and Friends to Expire, Reveal Crypto-Keys". In: *The Register* (Nov. 19, 2020). URL: https://www.theregister.com/2020/11/19/dkim_encryption_expiration/ (visited on 06/20/2021) (cit. on p. 19).

[133] Frederick R. Chang. *Is Your Data on the Healthcare.gov Website Secure?* Written Testimony, U.S. House of Representatives, Nov. 2013. URL: http://docs.house.gov/meetings/SY/SY00/20131119/101533/HHRG-113-SY00-Wstate-ChangF-20131119.pdf (cit. on p. 182).

[134] Oded Gal. *The Facts Around Zoom and Encryption for Meetings/Webinars*. Zoom Blog. Apr. 2, 2020. URL: https://blog.zoom.us/facts-around-zoom-encryption-for-meetings-webinars/ (visited on 03/28/2021) (cit. on p. 33).

[135] Sarah Gamard. *Delaware presidential primary: Here's how to vote from home on June 2 due to coronavirus*. en. Delaware News Journal. https://www.delawareonline.com/story/news/politics/2020/05/05/heres-how-vote-absentee-delaware-due-coronavirus/3048049001/. May 2020. (Visited on 05/12/2020) (cit. on p. 39).

[136] Julien Gamba, Mohammed Rashed, Abbas Razaghpanah, Juan Tapiador, and Narseo Vallina-Rodriguez. "An Analysis of Pre-Installed Android Software". In: *2020 IEEE Symposium on Security and Privacy (SP)*. May 2020, pp. 1039–1055. DOI: 10.1109/SP40000.2020.00013 (cit. on p. 13).

[137] Simson L. Garfinkel. "Carving contiguous and fragmented files with fast object validation". en. In: *Digital Investigation* 4 (Sept. 2007), pp. 2–12. ISSN: 1742-2876. DOI: 10.1016/j.diin.2007.06.017. (Visited on 05/21/2020) (cit. on p. 89).

[138] Pierrick Gaudry. "Breaking the encryption scheme of the Moscow internet voting system". In: *arXiv preprint arXiv:1908.05127* (2019). https://arxiv.org/pdf/1908.05127.pdf (cit. on pp. 39, 106).

[139] Gennie Gebhart and Eva Galperin. *FTC Takes Action Against Stalkerware Company Retina-X*. Electronic Frontier Foundation. Nov. 6, 2019. URL: https://www.eff.org/deeplinks/2019/11/ftc-takes-action-against-stalkerware-company-retina-x (visited on 08/27/2021) (cit. on p. 200).

[140] Eric Geller. *Coronavirus boosts push for online voting despite security risks*. en. Politico. https://www.politico.com/news/2020/05/01/coronavirus-online-voting-229690. May 2020. (Visited on 05/12/2020) (cit. on p. 39).

[141] Craig Gentry and Alice Silverberg. "Hierarchical ID-Based Cryptography". In: *Proceedings of ASIACRYPT 2002*. Ed. by Yuliang Zheng. Vol. 2501. Lecture Notes in Computer Science. Springer, 2002, pp. 548–566. ISBN: 3-540-00171-9. DOI: 10.1007/3-540-36178-2_34. URL: https://doi.org/10.1007/3-540-36178-2%5C_34 (cit. on pp. 135, 158).

[142] Richard George. *Views On the Future Direction of Information Assurance*. Remarks by Richard George at Blackhat Las Vegas. July 2002. URL: https://www.blackhat.com/presentations/bh-usa-02/bh-us-02-george-keynote.doc (cit. on p. 174).

[143] Micha Germann and Uwe Serdült. "Internet voting and turnout: Evidence from Switzerland". In: *Electoral Studies* 47 (2017), pp. 1–12 (cit. on p. 106).

[144] Dan Goodin. *FBI tells router users to reboot now to kill malware infecting 500k devices*. en-us. https://arstechnica.com/information-technology/2018/05/fbi-tells-router-users-to-reboot-now-to-kill-malware-infecting-500k-devices/. May 2018. (Visited on 01/20/2020) (cit. on p. 62).

[145] Dan Goodin. *Mass router hack exposes millions of devices to potent NSA exploit*. en-us. https://arstechnica.com/information-technology/2018/11/mass-router-hack-exposes-millions-of-devices-to-potent-nsa-exploit/. Nov. 2018. (Visited on 01/20/2020) (cit. on p. 62).

[146] Bob Goodlatte, Fred Upton, John Conyers Jr., Frank Pallone Jr., Jim Sensenbrenner, Darrell Issa, Zoe Lofgren, Suzan DelBene, Bill Johnson, and Yvette D. Clarke. *House Judiciary Committee & House Energy and Commerce Committee: Encryption Working Group Year-End Report*. Dec. 20, 2016. URL: https://republicans-energycommerce.house.gov/wp-content/uploads/2019/10/Encryption-Working-Group-Year-End-Report.pdf (cit. on pp. 19, 193).

[147] Google. *Google Analytics Terms of Service*. https://marketingplatform.google.com/about/analytics/terms/us/. June 2019 (cit. on p. 93).

[148] Google. *Making email safer for you*. en. Feb. 2016. URL: https://migration-dot-gweb-uniblog-publish-prod.appspot.com/products/gmail/making-email-safer-for-you-posted-by/ (visited on 11/10/2019) (cit. on p. 161).

[149] *Green Guides*. Federal Trade Commission. Oct. 31, 2018. URL: https://www.ftc.gov/news-events/media-resources/truth-advertising/green-guides%20[https://perma.cc/3VES-UEXK] (visited on 07/04/2021) (cit. on p. 32).

[150] Matthew Green. *A Few Thoughts on Ray Ozzie's "Clear" Proposal*. A Few Thoughts on Cryptographic Engineering. Apr. 26, 2018. URL: https://blog.cryptographyengineering.com/2018/04/26/a-few-thoughts-on-ray-ozzies-clear-proposal/ (visited on 08/18/2021) (cit. on p. 195).

[151] Matthew Green. *On Ghost Users and Messaging Backdoors*. A Few Thoughts on Cryptographic Engineering. Dec. 17, 2018. URL: https://blog.cryptographyengineering.com/2018/12/17/on-ghost-users-and-messaging-backdoors/ (visited on 08/18/2021) (cit. on p. 195).

[152] Matthew D. Green and Alex Stamos. "Opinion | Apple Wants to Protect Children. But It's Creating Serious Privacy Risks." In: *The New York Times. Opinion* (Aug. 11, 2021). ISSN: 0362-4331. URL: https://www.nytimes.com/2021/08/11/opinion/apple-iphones-privacy.html (visited on 08/18/2021) (cit. on p. 195).

[153] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *Advances in Cryptology – EUROCRYPT 2016*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 305–326. ISBN: 978-3-662-49896-5 (cit. on p. 150).

[154] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. "Message Franking via Committing Authenticated Encryption". In: *CRYPTO (3)*. Vol. 10403. Lecture Notes in Computer Science. Springer, 2017, pp. 66–97 (cit. on p. 195).

[155] Lachlan J. Gunn, Ricardo Vieitez Parra, and N. Asokan. *Circumventing Cryptographic Deniability with Remote Attestation*. 2019. DOI: 10.2478/popets-2019-0051. URL: https://doi.org/10.2478/popets-2019-0051 (cit. on p. 128).

[156] Deepak Gupta. *Google Enables 'Forward Secrecy (PFS)' by 'Default' for HTTPS Services*. Nov. 2011. URL: http://www.ditii.com/2011/11/23/google-enables-forward-secrecy-pfs-by-default-for-https-services/ (visited on 07/06/2015) (cit. on p. 177).

[157] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. "How not to prove your election outcome". In: *41st IEEE Symposium on Security and Privacy*. 2019 (cit. on pp. 39, 46, 104, 106).

[158] J. Alex Halderman. "Practical Attacks on Real-world E-voting". In: *Real-World Electronic Voting: Design, Analysis and Deployment*. Ed. by Feng Hao and Peter Y. A. Ryan. 2016, pp. 145–171 (cit. on p. 46).

[159] J. Alex Halderman and Vanessa Teague. "The New South Wales iVote System: Security Failures and Verification Flaws in a Live Online Election". In: *5th International Conference on E-voting and Identity*. E-VoteID. 2015 (cit. on pp. 39, 47).

[160] Harvie Branscomb. *Denver Voatz on cell phones – initial review*. http://electionquality.com/2019/05/denver-voatz-1/. May 2019. (Visited on 01/10/2020) (cit. on pp. 49, 61).

[161] Nelson Hastings, Rene Peralta, Stefan Popoveniuc, and Andrew Regenscheid. *Security Considerations for Remote Electronic UOCAVA Voting*. National Institute of Standards and Technology, NISTIR 7770. https://www.nist.gov/system/files/documents/itl/vote/NISTIR-7700-feb2011.pdf. 2011 (cit. on p. 84).

[162] *Hearing on "Deciphering the Debate Over Encryption: Industry and Law Enforcement Perspectives," Subcommittee on Oversight and Investigations (April 19, 2016)* (cit. on p. 193).

[163] Cormac Herley. "So Long, and No Thanks for the Externalities: The Rational Rejection of Security Advice by Users". In: *Proceedings of the 2009 Workshop on New Security Paradigms Workshop*. 2009, pp. 133–144 (cit. on pp. 14, 23).

[164] HIPAA Journal. *United Hospital District Phishing Attack Impacts 2,143 Patients*. https://www.hipaajournal.com/united-hospital-district-phishing-attack-impacts-2143-patients/. 2019 (cit. on p. 110).

[165] Chris Jay Hoofnagle. *Federal Trade Commission Privacy Law and Policy*. 2016 (cit. on p. 32).

[166] Chris Jay Hoofnagle, Aniket Kesari, and Aaron Perzanowski. "The Tethered Economy". In: *Geo. Wash. L. Rev.* 87 (2019), p. 783 (cit. on p. 24).

[167] *In Re BLU Products, Inc.* Sept. 6, 2018 (cit. on p. 34).

[168] *In Re Epic Marketplace, Inc.* Dec. 5, 2012 (cit. on p. 34).

[169] *In Re Henry Schein Practice Solutions, Inc.* Jan. 5, 2016 (cit. on p. 32).

[170] *In Re Snapchat, Inc.* Dec. 23, 2014 (cit. on pp. 32, 34).

[171] *In Re Zoom Video Comm'ns, Inc.* Jan. 19, 2021 (cit. on p. 33).

[172] *Investigatory Powers Act 2016*. 2016. URL: https://www.legislation.gov.uk/ukpga/2016/25/contents/enacted (visited on 08/18/2021) (cit. on p. 194).

[173] J. Levine. *RFC 8463 - A New Cryptographic Signature Method for DomainKeys Identified Mail (DKIM)*. URL: https://datatracker.ietf.org/doc/rfc8463/ (visited on 08/26/2019) (cit. on pp. 159, 161).

[174] Jaikumar Vijayan. *TJX data breach: At 45.6M card numbers, it's the biggest ever*. Mar. 2007. URL: http://www.computerworld.com/article/2544306/security0/tjx-data-breach--at-45-6m-card-numbers--it-s-the-biggest-ever.html (visited on 07/06/2015) (cit. on p. 173).

[175] Markus Jakobsson. "Fractal Hash Sequence Representation and Traversal". In: *Proceedings of the 2002 IEEE International Symposium on Information Theory (ISIT)* (2002), pp. 437–44 (cit. on p. 139).

[176] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. "Designated Verifier Proofs and Their Applications". In: *Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT'96. Saragossa, Spain: Springer-Verlag, 1996, pp. 143–154. ISBN: 3-540-61186-X. URL: http://dl.acm.org/citation.cfm?id=1754495.1754513 (cit. on p. 129).

[177] James B. Comey. *Going Dark: Are Technology, Privacy, and Public Safety on a Collision Course?* Speech at the Brookings Institution. Oct. 2014. URL: https://www.fbi.gov/news/speeches/going-dark-are-technology-privacy-and-public-safety-on-a-collision-course (visited on 07/06/2015) (cit. on p. 169).

[178] Jeanne Meserve. "Sources: Staged cyber attack reveals vulnerability in power grid - CNN.com". In: *CNN* (Sept. 2007). URL: http://www.cnn.com/2007/US/09/26/power.at.risk/index.html?iref=topnews (visited on 07/06/2015) (cit. on p. 174).

[179] Joseph Cox. *Instructions Show How Cops Use GrayKey to Brute Force iPhones*. June 22, 2021. URL: https://www.vice.com/en/article/k7835w/how-to-brute-force-iphones-graykey (visited on 08/18/2021) (cit. on p. 194).

[180] Joseph Cox. "The FBI's 'Unprecedented' Hacking Campaign Targeted Over a Thousand Computers". In: *Vice* (Jan. 5, 2016). URL: https://www.vice.com/en/article/qkj8vv/the-fbis-unprecedented-hacking-campaign-targeted-over-a-thousand-computers (visited on 08/18/2021) (cit. on p. 194).

[181] Seny Kamara. *Crypto for the People, Invited Talk at Crypto 2020 by Seny Kamara*. Annual International Cryptology Conference. Aug. 2020. URL: https://www.youtube.com/watch?v=Ygq9ci0GFhA (visited on 06/17/2021) (cit. on p. 15).

[182] Ruogu Kang, Laura Dabbish, Nathaniel Fruchter, and Sara Kiesler. ""My Data Just Goes Everywhere:" User Mental Models of the Internet and Implications for Privacy and Security". In: *Eleventh Symposium On Usable Privacy and Security (${$SOUPS$}$ 2015)*. 2015, pp. 39–52 (cit. on p. 23).

[183] Jamila Kaya and Jacob Rickerd. *Security Researchers Partner With Chrome To Take Down Browser Extension Fraud Network Affecting Millions of Users*. Duo Security Blog. https://duo.com/labs/research/crxcavator-malvertising-2020. 2020 (cit. on p. 84).

[184] Jemima Kiss. *Twitter adds more security to thwart predators – and government agencies*. Nov. 2013. URL: http://www.theguardian.com/technology/2013/nov/23/twitter-security-google-facebook-data-nsa (visited on 07/06/2015) (cit. on p. 177).

[185] John Klensin. *RFC5321: Simple Mail Transfer Protocol*. en. 2008. URL: https://tools.ietf.org/html/rfc5321 (visited on 04/05/2019) (cit. on pp. 116, 142).

[186] Amy Klobuchar. *S.1540 - 116th Congress (2019-2020): Election Security Act of 2019*. eng. webpage. https://www.congress.gov/bill/116th-congress/senate-bill/1540/text. May 2019. (Visited on 01/07/2020) (cit. on p. 40).

[187] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. "Analysis of an electronic voting system". In: *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*. IEEE, 2004, pp. 27–40 (cit. on p. 40).

[188] Brian Krebs. *Browser Extensions: Are They Worth the Risk?* Krebs on Security. https://krebsonsecurity.com/2018/09/browser-extensions-are-they-worth-the-risk/. Sept. 2018 (cit. on p. 84).

[189] Joshua A Kroll, Joanna Huey, Solon Barocas, Edward W Felten, Joel R Reidenberg, David G Robinson, and Harlan Yu. "Accountable Algorithms". In: *U. Penn. L. Rev.* 165 (2017), p. 633 (cit. on pp. 30, 162, 197).

[190] Murray Kucherawy and Elizabeth Zwicky. *Domain-based Message Authentication, Reporting, and Conformance (DMARC)*. en. https://tools.ietf.org/html/rfc7489. (Visited on 02/15/2019) (cit. on p. 117).

[191] Anunay Kulshrestha and Jonathan Mayer. "Identifying Harmful Media in End-to-End Encrypted Communication: Efficient Private Membership Computation". In: *30th ${$USENIX$}$ Security Symposium (${$USENIX$}$ Security 21)*. 2021 (cit. on p. 195).

[192] Kurt Andersen. *M3AAWG DKIM Key Rotation Best Common Practices | M3AAWG*. http://www.m3aawg.org/DKIMKeyRotation [https://perma.cc/4WY6-SH8K]. Mar. 2019 (cit. on p. 143).

[193] Susan Landau. *THE ENCRYPTION TIGHTROPE: BALANCING AMERICANS' SECURITY AND PRIVACY, Hearing before the Committee on the Judiciary, House of Representatives*. URL: https://docs.house.gov/meetings/JU/JU00/20160301/104573/HHRG-114-JU00-Transcript-20160301.pdf (cit. on p. 193).

[194] Larry Moore and Nimit Sawhney. *UNDER THE HOOD The West Virginia Mobile Voting Pilot*. https://www.nass.org/sites/default/files/2019-02/white-paper-voatz-nass-winter19.pdf. 2019 (cit. on p. 51).

[195] Larry Seltzer. *Google moves forward towards a more perfect SSL*. Nov. 2013. URL: http://www.zdnet.com/article/google-moves-forward-towards-a-more-perfect-ssl/ (visited on 07/06/2015) (cit. on p. 177).

[196] Ben Laurie, Adam Langley, and Emilia Kasper. *Certificate Transparency*. RFC 6962. June 2013. DOI: 10.17487/RFC6962. URL: https://rfc-editor.org/rfc/rfc6962.txt (cit. on p. 146).

[197] Micah Lee and Yael Grauer. *Zoom Meetings Aren't End-to-End Encrypted, Despite Misleading Marketing*. Mar. 2020. URL: https://theintercept.com/2020/03/31/zoom-meeting-encryption/ (cit. on p. 33).

[198] Samuel Levine. *Letter from Acting Director of the Bureau of Consumer Protection Samuel Levine to Facebook*. Letter. Aug. 5, 2021. URL: https://www.ftc.gov/news-events/blogs/consumer-blog/2021/08/letter-acting-director-bureau-consumer-protection-samuel (visited on 08/26/2021) (cit. on p. 199).

[199] Ian Levy and Crispin Robinson. *Principles for a More Informed Exceptional Access Debate*. Lawfare. Nov. 29, 2018. URL: https://www.lawfareblog.com/principles-more-informed-exceptional-access-debate (visited on 08/18/2021) (cit. on p. 195).

[200] Liam Tung. *Yahoo finally enables HTTPS encryption for email by default*. Jan. 2014. URL: http://www.zdnet.com/article/yahoo-finally-enables-https-encryption-for-email-by-default/ (visited on 07/06/2015) (cit. on p. 177).

[201] Einar Lielmanis. *beautify-web/js-beautify*. https://github.com/beautify-web/js-beautify. (Visited on 05/12/2020) (cit. on p. 72).

[202] Mark Lindeman and Philip B. Stark. "A gentle introduction to risk-limiting audits". In: *IEEE Security & Privacy* 10.5 (2012), pp. 42–49 (cit. on pp. 86, 104).

[203] Gavin Lowe. "An Attack on the Needham-Schroeder Public-key Authentication Protocol". In: *Information Processing Letters* 56.3 (Nov. 1995), pp. 131–133. ISSN: 0020-0190. DOI: 10.1016/0020-0190(95)00144-2. URL: http://dx.doi.org/10.1016/0020-0190(95)00144-2 (visited on 07/06/2015) (cit. on p. 190).

[204] Daniel Hays Lowenstein, Richard L. Hasen, Daniel P. Tokaji, and Nicholas Stephanopoulos. *Election Law: Cases and Materials*. 6th ed. See page 458 note (a) for quoted text. Carolina Academic Press, 2017 (cit. on p. 43).

[205] Sean Lyngaas. *HackerOne cuts ties with mobile voting firm Voatz after it clashed with researchers*. en. Mar. 2020. URL: https://www.cyberscoop.com/voatz-hackerone-bug-bounty-election-security/ (visited on 06/03/2020) (cit. on p. 102).

[206] *Magisk Manager*. en-US. https://magiskmanager.com/. (Visited on 01/13/2020) (cit. on p. 47).

[207] Tara Matthews, Kathleen O'Leary, Anna Turner, Manya Sleeper, Jill Palzkill Woelfer, Martin Shelton, Cori Manthorne, Elizabeth F. Churchill, and Sunny Consolvo. "Stories from survivors: Privacy & security practices when coping with intimate partner abuse". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 2017, pp. 2189–2201 (cit. on p. 71).

[208] Maya Kosoff. *"A Horrifically Bad Idea": Smartphone Voting is Coming, Just in Time for the Midterms*. https://www.vanityfair.com/news/2018/08/smartphone-voting-is-coming-just-in-time-for-midterms-voatz. Aug. 2018. (Visited on 01/10/2020) (cit. on p. 49).

[209] Jonathan Mayer. *Tracking the Trackers: To Catch a History Thief*. Stanford Ctr. for Internet & Society. July 19, 2011. URL: /blog/2011/07/tracking-trackers-catch-history-thief (visited on 05/05/2021) (cit. on p. 34).

[210] India McKinney and Erica Portnoy. *Apple's Plan to "Think Different" About Encryption Opens a Backdoor to Your Private Life*. Electronic Frontier Foundation. Aug. 5, 2021. URL: https://www.eff.org/deeplinks/2021/08/apples-plan-think-different-about-encryption-opens-backdoor-your-private-life (visited on 08/18/2021) (cit. on p. 195).

[211] Lucas Mearian. "Why blockchain-based voting could threaten democracy". en. In: *Computerworld* (Aug. 2019). https://www.computerworld.com/article/3430697/why-blockchain-could-be-a-threat-to-democracy.html. (Visited on 01/07/2020) (cit. on p. 41).

[212] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. "CONIKS: Bringing Key Transparency to End Users". In: *24th USENIX Security Symposium (USENIX Security 15)*. 2015, pp. 383–398 (cit. on p. 162).

[213] Jeremy B. Merrill. "Authenticating Email Using DKIM and ARC, or How We Analyzed the Kasowitz Emails". In: *ProPublica* (July 2017) (cit. on p. 109).

[214] Eran Messeri, Adam Langley, Emilia Kasper, Ben Laurie, and Rob Stradling. *Certificate Transparency*. July 6, 2015. URL: https://tools.ietf.org/html/draft-ietf-trans-rfc6962-bis-08 (visited on 09/16/2015) (cit. on p. 162).

[215] Michael Mimoso. *Microsoft Brings Perfect Forward Secrecy to Windows | Threatpost | The first stop for security news*. May 2015. URL: https://threatpost.com/new-crypto-suites-bring-perfect-forward-secrecy-to-windows/112783 (visited on 07/06/2015) (cit. on p. 177).

[216] Michael Mimoso. *Microsoft Expands TLS, Forward Secrecy Support | Threatpost | The first stop for security news*. July 2014. URL: https://threatpost.com/microsoft-expands-tls-forward-secrecy-support/106965 (visited on 07/06/2015) (cit. on p. 177).

[217] Microsoft Defending Democracy Program. *ElectionGaurd*. https://github.com/microsoft/electionguard. 2019 (cit. on p. 46).

[218] Mobile Voting Project. *New Jersey Announces Accessible Voting is Coming to May Elections*. https://mobilevoting.org/2020/05/new-jersey-announces-accessible-voting-is-coming-to-may-elections/. May 2020 (cit. on p. 76).

[219] Mobile Voting Project. *West Virginia Expands Online Voting Option in Upcoming Primary Election for Citizens with Disabilities*. https://mobilevoting.org/2020/04/west-virginia-expands-online-voting-option-in-upcoming-primary-election-for-citizens-with-disabilities/. Apr. 2020 (cit. on p. 39).

[220] Tyler Moore. "The Economics of Cybersecurity: Principles and Policy Options". In: *International Journal of Critical Infrastructure Protection* 3.3-4 (2010), pp. 103–117 (cit. on p. 23).

[221] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Tech. rep. Manubot, 2019 (cit. on p. 40).

[222] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2009. URL: http://www.bitcoin.org/bitcoin.pdf (cit. on p. 146).

[223] Moni Naor. "Deniable Ring Authentication". In: *Advances in Cryptology — CRYPTO 2002*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 481–498. ISBN: 978-3-540-45708-4 (cit. on p. 129).

[224] National Academies of Sciences, Engineering, and Medicine. *Securing the Vote: Protecting American Democracy*. Washington, DC: The National Academies Press, 2018. ISBN: 978-0-309-47647-8. DOI: 10.17226/25120. URL: https://www.nap.edu/catalog/25120/securing-the-vote-protecting-american-democracy (cit. on pp. 39, 102, 106).

[225] National Cybersecurity Center. *NCC King County Audit Summary 2020*. https://cyber-center.org/ncc-king-county-audit-summary-2020/. Mar. 2020 (cit. on p. 39).

[226] Roger M. Needham and Michael D. Schroeder. "Using encryption for authentication in large networks of computers". In: *Communications of the ACM* 21.12 (1978), pp. 993–999. URL: http://dl.acm.org/citation.cfm?id=359659 (visited on 07/06/2015) (cit. on p. 190).

[227] New York County District Attorney. *REPORT OF THE MANHATTAN DISTRICT ATTORNEY'S OFFICE ON SMARTPHONE ENCRYPTION AND PUBLIC SAFETY*. Nov. 2015. URL: https://www.manhattanda.org/wp-content/themes/dany/files/11.18.15%5C%20Report%5C%20on%5C%20Smartphone%5C%20Encryption%5C%20and%5C%20Public%5C%20Safety.pdf (cit. on p. 193).

[228] *News: Android Firmware Sending Private Information Without Consent*. Kryptowire. URL: https://www.kryptowire.com/kryptowire-discovers-mobile-phone-firmware-transmitted-personally-identifiable-information-pii-without-user-consent-disclosure/ (visited on 05/05/2021) (cit. on p. 34).

[229] Robert W. Ney. *H.R.3295 - 107th Congress (2001-2002): Help America Vote Act of 2002*. eng. webpage. https://www.congress.gov/bill/107th-congress/house-bill/3295. Oct. 2002. (Visited on 01/07/2020) (cit. on p. 40).

[230] Nicole Perlroth and Vindu Goel. *Twitter Toughening Its Security to Thwart Government Snoops*. Nov. 2013. URL: http://bits.blogs.nytimes.com/2013/11/22/twitter-toughening-its-security-to-thwart-government-snoops/ (visited on 07/06/2015) (cit. on p. 177).

[231] NIST. *NIST Randomness Beacon*. https://www.nist.gov/programs-projects/nist-randomness-beacon (cit. on p. 146).

[232] NTIA Safety Working Group. *"Early Stage" Coordinated Vulnerability Disclosure Template*. https://www.ntia.doc.gov/files/ntia/publications/ntia_vuln_disclosure_early_stage_template.pdf. 2016 (cit. on p. 105).

[233] Emad Omara, Benjamin Beurdouche, Jon Millican, Raphael Robert, Katriel Cohn-Gordon, and Richard Barnes. *The Messaging Layer Security (MLS) Protocol*. en. https://tools.ietf.org/html/draft-ietf-mls-protocol-07 [https://perma.cc/YQ5X-36LB]. (Visited on 08/27/2019) (cit. on p. 201).

[234] Open Whisper Systems. *Curve.java*. https://perma.cc/6CBK-745E. 2018 (cit. on p. 201).

[235] Raymond Edward Ozzie. "Providing Low Risk Exceptional Access". U.S. pat. 20170272248A1. Raymond Edward Ozzie. Sept. 21, 2017. URL: https://patents.google.com/patent/US20170272248A1/en?q=cryptographic&inventor=ozzie&before=priority:20180101&after=priority:20140101&oq=cryptographic+inventor:ozzie+before:priority:20180101+after:priority:20140101 (visited on 08/16/2021) (cit. on p. 194).

[236] Sunoo Park, Michael Specter, Neha Narula, and Ronald L. Rivest. "Going from Bad to Worse: From Internet Voting to Blockchain Voting". In: *Journal of Cybersecurity* 7.1 (2021), tyaa025 (cit. on pp. 17, 106).

[237] Parker Higgins. *Pushing for Perfect Forward Secrecy, an Important Web Privacy Protection*. Aug. 2013. URL: https://www.eff.org/deeplinks/2013/08/pushing-perfect-forward-secrecy-important-web-privacy-protection (visited on 07/06/2015) (cit. on p. 177).

[238] Miles Parks. *States Expand Internet Voting Experiments Amid Pandemic, Raising Security Fears*. en. NPR News. https://www.npr.org/2020/04/28/844581667/states-expand-internet-voting-experiments-amid-pandemic-raising-security-fears. Apr. 2020. (Visited on 05/12/2020) (cit. on p. 73).

[239] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. "Pinocchio: Nearly Practical Verifiable Computation". In: *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, May 2013. URL: https://www.microsoft.com/en-us/research/publication/pinocchio-nearly-practical-verifiable-computation/ (cit. on p. 150).

[240] Patrick Leahy. *Statement of Senator Patrick Leahy (D-Vt.), Ranking Member, Senate Judiciary Committee, Hearing on "Going Dark: Encryption, Technology, and the Balance Between Public Safety and Privacy"*. URL: https://www.judiciary.senate.gov/meetings/going-dark-encryption-technology-and-the-balance-between-public-safety-and-privacy (visited on 08/17/2021) (cit. on p. 193).

[241] Paul Ford. *The Obamacare Website Didn't Have to Fail. How to Do Better Next Time*. Oct. 2013. URL: %5Curl%7Bhttp://www.bloomberg.com/bw/articles/2013-10-16/open-source-everything-the-moral-of-the-healthcare-dot-gov-debacle%7D (visited on 07/06/2015) (cit. on p. 173).

[242] Nicole Perlroth. "Electronic Security a Worry in an Age of Digital Espionage". In: *The New York Times* (Feb. 2012). ISSN: 0362-4331. URL: http://www.nytimes.com/2012/02/11/technology/electronic-security-a-worry-in-an-age-of-digital-espionage.html (visited on 07/06/2015) (cit. on p. 181).

[243] Nicole Perlroth. "Security Experts Oppose Government Access to Encrypted Communication". In: *The New York Times. Technology* (July 7, 2015). ISSN: 0362-4331. URL: https://www.nytimes.com/2015/07/08/technology/code-specialists-oppose-us-and-british-government-access-to-encrypted-communication.html (visited on 08/14/2021) (cit. on p. 192).

[244] Adrian Perrig, Ran Canetti, J.D. Tygar, and Dawn Song. "The TESLA Broadcast Authentication Protocol". In: *RSA CryptoBytes* 5 (2002), pp. 2–13 (cit. on pp. 139, 203).

[245] Adrian Perrig, Dawn Song, Ran Canetti, J. D. Tygar, and Bob Briscoe. "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction". In: *RFC* 4082 (2005), pp. 1–22. DOI: 10.17487/RFC4082. URL: https://doi.org/10.17487/RFC4082 (cit. on pp. 139, 203).

[246] Yngve N. Pettersen. *The Transport Layer Security (TLS) Multiple Certificate Status Request Extension*. RFC 6961. June 2013. DOI: 10.17487/RFC6961. URL: https://rfc-editor.org/rfc/rfc6961.txt (cit. on p. 145).

[247] Piero Colaprico. *"Da Telecom dossier sui Ds" Mancini parla dei politici - cronaca - Repubblica.it*. Jan. 2007. URL: http://www.repubblica.it/2006/12/sezioni/cronaca/sismi-mancini-8/dossier-ds/dossier-ds.html (visited on 07/06/2015) (cit. on p. 183).

[248] Krzysztof Pietrzak. "Simple Verifiable Delay Functions". In: *IACR Cryptology ePrint Archive* 2018 (2018), p. 627 (cit. on p. 146).

[249] Vassilis Prevelakis and Diomidis Spinellis. "The Athens Affair". In: *Spectrum, IEEE* 44.7 (2007), pp. 26–33. URL: http://ieeexplore.ieee.org/xpls/abs%5C_all.jsp?arnumber=4263124 (visited on 07/05/2015) (cit. on pp. 174, 183).

[250] Sheri Pym. "Order Compelling Apple, Inc. To Assist Agents in Search". In: *United States District Court for the Central District of California* (2016). URL: https://www.justice.gov/usao-cdca/file/825001/download (cit. on p. 194).

[251] J. Alex Halderman Rachel Goodman. *Internet Voting Is Happening Now And it could destroy our elections.* en. https://slate.com/technology/2020/01/internet-voting-could-destroy-our-elections.html. Jan. 2020. (Visited on 01/19/2020) (cit. on p. 106).

[252] Sara Randazzo. "Legal Heavyweights Olson, Boutrous Tapped to Help Apple in Encryption Fight". In: *Wall Street Journal. Tech* (Feb. 19, 2016). ISSN: 0099-9660. URL: https://www.wsj.com/articles/legal-heavyweights-olson-boutrous-tapped-to-help-apple-in-encryption-fight-1455906302 (visited on 08/18/2021) (cit. on p. 194).

[253] Eric S. Raymond, ed. *The New Hacker's Dictionary - 3rd Edition.* third edition. Cambridge, Mass: The MIT Press, Oct. 11, 1996. 547 pp. ISBN: 978-0-262-68092-9 (cit. on p. 24).

[254] *Realm.* https://realm.io/. (Visited on 01/19/2020) (cit. on p. 57).

[255] Rainey Reitman. *Deep Dive: Updating the Electronic Communications Privacy Act.* Electronic Frontier Foundation. Dec. 6, 2012. URL: https://www.eff.org/deeplinks/2012/12/deep-dive-updating-electronic-communications-privacy-act (visited on 08/27/2021) (cit. on p. 200).

[256] Richard George. *Private communication between Richard George, Former Technical Director, Information Assurance Directorate, NSA and Susan Landau.* Dec. 2011 (cit. on p. 175).

[257] R. L. Rivest, A. Shamir, and D. A. Wagner. *Time-lock Puzzles and Timed-release Crypto.* Tech. rep. Cambridge, MA, USA, 1996 (cit. on p. 146).

[258] Ronald L Rivest. "On the notion of 'software independence' in voting systems". In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 366.1881 (2008), pp. 3759–3767. ISSN: 1364-503X. DOI: 10.1098/rsta.2008.0149. eprint: http://rsta.royalsocietypublishing.org/content/366/1881/3759.full.pdf. URL: http://rsta.royalsocietypublishing.org/content/366/1881/3759 (cit. on pp. 43, 45).

[259] Ronald L. Rivest, Adi Shamir, and Yael Tauman. "How to Leak a Secret". In: *ASIACRYPT.* 2001 (cit. on p. 129).

[260] Ronald L. Rivest and Madars Virza. "Software Independence Revisited". In: *Real-World Electronic Voting: Design, Analysis and Deployment.* Ed. by Feng Hao and Peter Y. A. Ryan. Taylor & Francis. Chap. 1 (cit. on p. 43).

[261] David G. Robinson and J. Alex Halderman. "Ethical Issues in E-Voting Security Analysis". In: *2nd Workshop on Ethics in Computer Security Research.* WECSR. 2011 (cit. on p. 47).

[262] Phillip Rogaway. "The Moral Character of Cryptographic Work". In: Austin, TX: USENIX Association, Aug. 2016 (cit. on pp. 15, 197).

[263] Ron Wyden. *Sen. Ron Wyden (D-Ore.) Letter Regarding Voatz.* https://www.washingtonpost.com/context/sen-ron-wyden-d-ore-letter-regarding-voatz/e9e6dd4f-1752-4c46-8e37-08a0f21dd042/. Nov. 2019 (cit. on p. 50).

[264] Matthew Rosenberg. *Voting on Your Phone: New Elections App Ignites Security Debate*. URL: https://www.nytimes.com/2020/02/13/us/politics/voting-smartphone-app.html (visited on 06/03/2020) (cit. on p. 101).

[265] Peter YA Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. "Prêt à voter: a voter-verifiable voting system". In: *IEEE transactions on information forensics and security* 4.4 (2009), pp. 662–673 (cit. on p. 46).

[266] Pamela Samuelson and Suzanne Scotchmer. "The Law and Economics of Reverse Engineering". en. In: *The Yale Law Journal* 111.7 (May 2002), p. 1575. ISSN: 00440094. DOI: 10.2307/797533. URL: https://www.jstor.org/stable/797533?origin=crossref (visited on 06/22/2020) (cit. on pp. 21, 22).

[267] David E. Sanger. *A Botnet Is Taken Down in an Operation by Microsoft, Not the Government*. The New York Times. https://www.nytimes.com/2020/03/10/us/politics/microsoft-botnets-malware.html. Mar. 2020 (cit. on p. 84).

[268] Raphael Satter. "Emails: Lawyer who met Trump Jr. tied to Russian officials". In: *The Associated Press* (July 2018) (cit. on p. 109).

[269] Stefan Savage. "Lawful Device Access without Mass Surveillance Risk: A Technical Design Discussion". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 1761–1774 (cit. on p. 194).

[270] *Savecrypto.Org*. Oct. 13, 2015. URL: http://web.archive.org/web/20151013154551/https://savecrypto.org/ (visited on 08/17/2021) (cit. on p. 193).

[271] Sophia Schmidt. *Delaware drops Internet-based voting system used by some absentee voters amid security concerns*. Delaware Public Media. https://www.delawarepublic.org/post/delaware-drops-internet-based-voting-system-used-some-absentee-voters-amid-security-concerns. June 2020 (cit. on p. 101).

[272] Selena Larson. *After Heartbleed, "Forward Secrecy" Is More Important Than Ever*. Apr. 2014. URL: http://readwrite.com/2014/04/15/heartbleed-perfect-forward-secrecy-security-encryption (visited on 07/06/2015) (cit. on p. 177).

[273] Adi Shamir. "Identity-Based Cryptosystems and Signature Schemes". In: *Proceedings of CRYPTO '84*. Ed. by G. R. Blakley and David Chaum. Vol. 196. Lecture Notes in Computer Science. Springer, 1984, pp. 47–53. ISBN: 3-540-15658-5. DOI: 10.1007/3-540-39568-7_5. URL: https://doi.org/10.1007/3-540-39568-7%5C_5 (cit. on p. 133).

[274] skylot. *skylot/jadx*. https://github.com/skylot/jadx. Jan. 2020. (Visited on 01/13/2020) (cit. on p. 59).

[275] SLI Compliance. *Democracy Live Secure Select 1.0 California Certification Security and Telecommunications Test Report*. https://votingsystems.cdn.sos.ca.gov/vendors/demlive/sli-dl-sectel.pdf. 2017 (cit. on p. 103).

[276] Daniel J Solove and Woodrow Hartzog. "The FTC and the new common law of privacy". In: *Colum. L. Rev.* 114 (2014), p. 583 (cit. on p. 98).

[277] Michael A. Specter, James Koppel, and Daniel Weitzner. "The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in U.S. Federal Elections". In: *29th USENIX Security Symposium*. Aug. 2020 (cit. on pp. 14, 17, 39).

[278] Michael A. Specter, Sunoo Park, and Matthew Green. "KeyForge: Non-Attributable Email from Forward-Forgeable Signatures". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/specter-keyforge (cit. on p. 14).

[279] Michael Specter and Alex Halderman. "Security Analysis of the Democracy Live Online Voting System". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/specter-security (cit. on pp. 14, 17).

[280] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J. Alex Halderman. "Security analysis of the Estonian internet voting system". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 703–715 (cit. on pp. 39, 66, 106).

[281] Philip B Stark and David Wagner. "Evidence-based elections". In: *IEEE Security & Privacy* 10.5 (2012), pp. 33–41 (cit. on pp. 42, 45).

[282] Tim Starks. *States dabble with online voting*. en. Politico. https://www.politico.com/newsletters/morning-cybersecurity/2020/04/30/states-dabble-with-online-voting-787248. Apr. 2020. (Visited on 05/12/2020) (cit. on p. 76).

[283] State of Delaware. *Egress User Guide for External Users*. https://elections.delaware.gov/information/pdfs/Egress%20Guide%20for%20External%20Users.pdf (cit. on p. 90).

[284] *Stay Secure with Jumio's Certified 3D Liveness Detection*. https://www.jumio.com/about/press-releases/3d-liveness-detection/. Oct. 2018. (Visited on 01/10/2020) (cit. on p. 55).

[285] Jonathan Stempel and Jim Finkle. *Yahoo says all three billion accounts hacked in 2013 data theft*. https://www.reuters.com/article/us-yahoo-cyber/yahoo-says-all-three-billion-accounts-hacked-in-2013-data-theft-idUSKCN1C82O1. 2017 (cit. on p. 110).

[286] Steven Rosenfeld. *Counting Voatz: Inside America's Most Radical Voting Technology*. https://www.nationalmemo.com/counting-voatz-inside-americas-most-radical-voting-technology/. May 2019. (Visited on 01/08/2020) (cit. on p. 49).

[287] Center for Strategic and International Studies (CSIS). *Significant Cyber Incidents*. https://www.csis.org/programs/cybersecurity-and-governance/technology-policy-program/other-projects-cybersecurity. 2018 (cit. on p. 110).

[288] Peter Swire. *Testimony of Peter Swire, Going Dark: Encryption, Technology, and the Balance Between Public Safety and Privacy | United States Senate Committee on the Judiciary*. URL: https://www.judiciary.senate.gov/imo/media/doc/07-08-15%5C%20Swire%5C%20Testimony.pdf (visited on 08/17/2021) (cit. on p. 193).

[289] *Telecommunications and Other Legislation Amendment (Assistance and Access) Act 2018*. 2018. URL: http://www.legislation.gov.au/Details/C2018A00148 (visited on 08/18/2021) (cit. on p. 194).

[290] Michael Thomas. *Requirements for a DomainKeys Identified Mail (DKIM) Signing Practices Protocol*. en. https://tools.ietf.org/html/rfc5016. (Visited on 02/15/2019) (cit. on p. 117).

[291] Bennie G. Thompson. *H.R.2660 - 116th Congress (2019-2020): Election Security Act of 2019*. eng. webpage. https://www.congress.gov/bill/116th-congress/house-bill/2660/text. June 2019. (Visited on 01/07/2020) (cit. on p. 40).

[292] Nathan Thornburgh. "The Invasion of the Chinese Cyberspies". In: *Time* (Aug. 2005). ISSN: 0040-718X. URL: http://content.time.com/time/magazine/article/0,9171,1098961,00.html (visited on 07/06/2015) (cit. on p. 173).

[293] Tom Cross. *Exploring Lawful Intercept to Wiretap the Internet*. Washington, DC, USA, 2010. URL: https://www.blackhat.com/presentations/bh-dc-10/Cross%5C_Tom/BlackHat-DC-2010-Cross-Attacking-LawfulI-Intercept-slides.pdf (cit. on p. 174).

[294] Trend Micro. *Hacker Infects Node.js Package to Steal from Bitcoin Wallets*. https://www.trendmicro.com/vinfo/hk-en/security/news/cybercrime-and-digital-threats/hacker-infects-node-js-package-to-steal-from-bitcoin-wallets. Nov. 2018 (cit. on p. 84).

[295] *Trillian: General Transparency*. Google, Aug. 24, 2021. URL: https://github.com/google/trillian (visited on 08/25/2021) (cit. on p. 162).

[296] U.S. Census Bureau. *2017 FIPS Codes*. https://www.census.gov/geographies/reference-files/2017/demo/popest/2017-fips.html. 2017 (cit. on p. 76).

[297] U.S. Cybersecurity and Infrastructure Security Agency. *Risk Management for Electronic Ballot Delivery, Marking, and Return (Draft)*. Published by The Guardian. https://www.scribd.com/document/460491458/CISA-Guidelines-on-Internet-Voting. May 2020 (cit. on p. 93).

[298] U.S. Election Assistance Commission. *A Survey of Internet Voting*. https://www.eac.gov/sites/default/files/eac_assets/1/28/SIV-FINAL.pdf. 2011 (cit. on p. 73).

[299] U.S. Election Assistance Commission Technical Guidelines Development Committee. *Recommendations for Requirements for the Voluntary Voting System Guidelines 2.0*. https://www.eac.gov/sites/default/files/TestingCertification/2020_02_29_vvsg_2_draft_requirements.pdf. Feb. 2020 (cit. on pp. 41, 104).

[300] U.S. Senate Select Committee on Intelligence. *Russian Active Measure Campaigns and Interference in the 2016 U.S. Election, Volume 1: Russian Efforts Against Election Infrastructure*. https://www.intelligence.senate.gov/sites/default/files/documents/Report_Volume1.pdf. 2019 (cit. on p. 39).

[301] Nik Unger and Ian Goldberg. "Deniable Key Exchanges for Secure Messaging". In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. Denver, Colorado, USA: ACM, 2015, pp. 1211–1223. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813616. URL: http://doi.acm.org/10.1145/2810103.2813616 (cit. on pp. 118, 119).

[302] Nik Unger and Ian Goldberg. "Improved Strongly Deniable Authenticated Key Exchanges for Secure Messaging". In: *PoPETs* 2018.1 (2018), pp. 21–66. DOI: 10.1515/popets-2018-0003. URL: https://doi.org/10.1515/popets-2018-0003 (cit. on pp. 109, 118).

[303] Keyon Vafa, Christian Haigh, Alvin Leung, and Noah Yonack. "Price Discrimination in The Princeton Review's Online SAT Tutoring Service". In: *Technology Science* (Aug. 31, 2015). URL: https://techscience.org/a/2015090102/ (visited on 05/08/2021) (cit. on p. 34).

[304] Valve. *Fingerprint.js*. https://github.com/Valve/fingerprintjs2. (Visited on 05/17/2020) (cit. on p. 83).

[305] Rory Van Loo. "The Missing Regulatory State: Monitoring Businesses in an Age of Surveillance". In: *Vanderbilt Law Review* 72 (2019), pp. 1563–1631 (cit. on p. 32).

[306] Voatz. *Voatz Response to Researchers' Flawed Report*. en-US. https://blog.voatz.com/?p=1209. Feb. 2020. (Visited on 06/08/2020) (cit. on p. 101).

[307] *Voatz - Bug Bounty Program*. en. https://hackerone.com/voatz (Since deactivated). (Visited on 01/21/2020) (cit. on p. 95).

[308] *Voatz FAQ*. https://voatz.com/faq.html [https://perma.cc/FBQ8-N875]. (Visited on 01/07/2020) (cit. on pp. 48, 95).

[309] *Voatz Open Press Call Transcribed from February 13, 2020*. en-US. Feb. 2020. URL: https://blog.voatz.com/?p=1243 (visited on 06/09/2020) (cit. on p. 101).

[310] *Voatz Security Issue Disclosure Policy*. en-US. https://blog.voatz.com/?p=1278. Aug. 2018. (Visited on 06/01/2020) (cit. on p. 96).

[311] Dustin Volz. *Agencies Warn States That Internet Voting Poses Widespread Security Risks*. The Wall Street Journal. https://www.wsj.com/articles/agencies-warn-states-that-internet-voting-poses-widespread-security-risks-11588975848. May 2020 (cit. on pp. 39, 93).

[312] Washington Post Editors. *7th Republican Debate Transcript, Annotated: Who Said What and What It Meant - The Washington Post*. Jan. 28, 2016. URL: https://www.washingtonpost.com/news/the-fix/wp/2016/01/28/7th-republican-debate-transcript-annotated-who-said-what-and-what-it-meant/ (visited on 08/17/2021) (cit. on p. 193).

[313] Mark Weiser. "The Computer for the 21st Century". In: *ACM SIGMOBILE mobile computing and communications review* 3.3 (1999), pp. 3–11 (cit. on p. 13).

[314] Mitchell Weiss and Maddy Halyard. *Voatz*. Case Study. Harvard Business Review, 2019 (cit. on pp. 49, 105).

[315] *What Is Free Software? - GNU Project - Free Software Foundation*. URL: https://www.gnu.org/philosophy/free-sw.en.html (visited on 08/27/2021) (cit. on p. 30).

[316] WhatsApp. *WhatsApp Encryption Overview: Technical White Paper*. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf [https://perma.cc/6L5W-C8S5] (cit. on p. 201).

[317] Alma Whitten and J. Doug Tygar. "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0." In: *USENIX Security Symposium*. Vol. 348. 1999, pp. 169–184 (cit. on p. 23).

[318] Wikileaks. *Search the DNC Database*. https://wikileaks.org/dnc-emails/. July 2016 (cit. on p. 109).

[319] Wikileaks. *Sony Email Leak*. https://wikileaks.org/sony/emails/. 2012 (cit. on p. 110).

[320] Wikileaks. *The Global Intelligence Files: STRATFOR email leak*. https://wikileaks.org/gifiles/docs/13/1328496_stratfor-.html. 2012 (cit. on p. 110).

[321] Wikileaks. *WikiLeaks: DKIM Verification*. https://wikileaks.org/DKIM-Verification.html [https://perma.cc/H3SR-YB44]. Nov. 2016. (Visited on 01/30/2019) (cit. on pp. 109, 142, 161).

[322] William J. Lynn III. "Defending a New Domain". In: *Foreign Affairs* (Oct. 2010). URL: https://www.foreignaffairs.com/articles/united-states/2010-09-01/defending-new-domain (visited on 07/06/2015) (cit. on p. 173).

[323] Scott Wolchok, Eric Wustrow, Dawn Isabel, and J. Alex Halderman. "Attacking the Washington, D.C. Internet Voting System". In: *16th Intl. Conference on Financial Cryptography and Data Security*. FC. 2012 (cit. on pp. 39, 76, 104).

[324] Gavin Wood. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151 (2014), pp. 1–32 (cit. on p. 146).

[325] Woodrow Hartzog and Daniel J. Solove. "The Scope and Potential of FTC Data Protection". In: *Geo. Wash. L. Rev.* 83 (2015), p. 2230 (cit. on p. 32).

[326] Charles Wright and Mayank Varia. "Crypto Crumple Zones: Enabling Limited Access without Mass Surveillance". In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 288–306 (cit. on p. 194).

[327] Ting-Fang Yen, Yinglian Xie, Fang Yu, Roger Peng Yu, and Martin Abadi. "Host Fingerprinting and Tracking on the Web: Privacy and Security Implications". In: *19th Network and Distributed System Security Symposium*. NDSS. 2012 (cit. on p. 92).

[328] Filip Zagórski, Richard T. Carback, David Chaum, Jeremy Clark, Aleksander Essex, and Poorvi L. Vora. "Remotegrity: Design and use of an end-to-end verifiable remote voting system". In: *International Conference on Applied Cryptography and Network Security*. Springer, 2013, pp. 441–457 (cit. on p. 71).

[329] Kim Zetter. *A Cyberattack Has Caused Confirmed Physical Damage for the Second Time Ever*. Jan. 2015. URL: http://www.wired.com/2015/01/german-steel-mill-hack-destruction/ (visited on 07/06/2015) (cit. on p. 174).

[330] Kim Zetter. *Google Hack Attack Was Ultra Sophisticated, New Details Show*. Wired. https://www.wired.com/2010/01/operation-aurora/. Jan. 2010 (cit. on p. 85).

[331] Kim Zetter. "Virginia Finally Drops America's 'Worst Voting Machines'". In: *Wired* (Aug. 2015). https://www.wired.com/2015/08/virginia-finally-drops-americas-worst-voting-machines/. ISSN: 1059-1028. (Visited on 01/12/2020) (cit. on p. 40).